目　录

第一章　概论 …… (1)
1.1　MATLAB 软件包的特征 …… (1)
1.1.1　MATLAB 语言（软件包）的发展过程 …… (1)
1.1.2　MATLAB 的功能 …… (2)
1.1.3　MATLAB 的工具箱 …… (2)
1.2　MATLAB 的系统需求和安装 …… (3)
1.2.1　MATLAB 的系统需求 …… (3)
1.2.2　MATLAB 的安装 …… (4)
1.3　神经网络发展和应用 …… (7)
1.3.1　人工神经网络发展的历史回顾 …… (7)
1.3.2　神经网络的应用 …… (9)
1.3.3　神经网络的学习方法 …… (9)
1.4　面向 MATLAB 工具箱的神经网络设计概述 …… (11)
1.4.1　MATLAB 神经网络工具箱 …… (11)
1.4.2　神经网络技术的选取 …… (12)
1.4.3　运用工具箱设计网络的原则和过程 …… (12)
1.5　本书的基本结构和内容概要 …… (13)
第二章　MATLAB 数值计算功能 …… (15)
2.1　矩阵与数组运算 …… (15)
2.1.1　矩阵的建立 …… (15)
2.1.2　矩阵和数组运算指令对照汇总 …… (18)
2.2　矩阵与数组函数 …… (21)
2.2.1　基本数组函数 …… (21)
2.2.2　基本矩阵函数 …… (22)
2.2.3　几个易混淆的两种函数运算 …… (24)
2.3　关系运算和逻辑运算 …… (25)
2.3.1　关系运算 …… (26)
2.3.2　逻辑运算 …… (26)
2.4　矩阵的分解 …… (28)
2.4.1　三角分解 …… (28)
2.4.2　正交分解 …… (29)
2.4.3　特征值分解 …… (29)
2.4.4　奇异值分解 …… (30)
2.5　多项式 …… (31)

2.5.1　多项式的表达和创建 …… (31)

2.5.2　多项式的运算 …… (31)

2.6　数据分析 …… (33)

2.6.1　基本统计函数指令 …… (33)

2.6.2　协方差阵和相关阵 …… (34)

2.6.3　有限差分和导数 …… (34)

2.6.4　数据滤波 …… (37)

2.7　数值分析 …… (37)

2.7.1　数值积分 …… (37)

2.7.2　微分方程的数值解 …… (39)

第三章　MATLAB 符号处理 …… (41)

3.1　字符串 …… (43)

3.1.1　字符数组 …… (44)

3.1.2　字符的 ASCII 码转换 …… (44)

3.1.3　创建二维的字符数组 …… (45)

3.1.4　字符串中的单元数组 …… (45)

3.1.5　字符数组与单元数组间的转换 …… (46)

3.1.6　字符串比较 …… (46)

3.1.7　判断字符串是否相等 …… (46)

3.1.8　通过字符的运算来比较字符 …… (47)

3.1.9　字符串中字符的分类 …… (47)

3.1.10　查找与替换 …… (48)

3.1.11　字符串和数值的相互转换 …… (48)

3.2　符号矩阵的运算 …… (49)

3.2.1　符号矩阵的创建 …… (49)

3.2.2　符号矩阵的加、减、乘和除运算 …… (50)

3.2.3　符号矩阵的逆和除运算 …… (51)

3.2.4　符号矩阵的幂运算 …… (51)

3.2.5　符号矩阵的综合运算指令 …… (52)

3.2.6　符号变量替换 …… (52)

3.2.7　符号矩阵的分解 …… (53)

3.2.8　符号微积分 …… (54)

3.2.9　符号代数方程的求解 …… (55)

3.2.10　符号微分方程的求解 …… (56)

第四章　绘图 …… (58)

4.1　二维绘图 …… (58)

4.1.1　plot …… (58)

4.1.2　figure 和 subplot …… (60)

4.1.3　绘图指令的开关控制 …… (62)

4.1.4 标题与坐标轴的操作 …… (65)
4.2 三维绘图 …… (68)
4.2.1 mesh …… (68)
4.2.2 3D 图形的颜色、光线来源及图上标点的设定 …… (70)
4.2.3 透视与视角的设置 …… (72)
4.3 图形句柄 …… (74)
4.3.1 图形对象 …… (74)
4.3.2 图形对象的句柄 …… (75)
4.3.3 对象创建函数 …… (76)
4.3.4 对象品性及其设置和查询 …… (85)
4.3.5 实时动画的制作 …… (85)
第五章 MATLAB 的程序设计 …… (88)
5.1 MATLAB 程序设计入门 …… (88)
5.1.1 编辑程序和 m 文件的形式 …… (88)
5.1.2 MATLAB 的命令文件 …… (89)
5.1.3 MATLAB 的函数文件 …… (90)
5.2 参数与变量 …… (91)
5.2.1 参数 …… (91)
5.2.2 局部变量与全局变量 …… (94)
5.3 数据类型 …… (96)
5.4 程序结构 …… (97)
5.4.1 顺序结构 …… (97)
5.4.2 循环结构 …… (98)
5.4.3 分支结构 …… (99)
5.5 程序流控制语句 …… (102)
5.5.1 echo 指令 …… (102)
5.5.2 input、yesinput 指令 …… (103)
5.5.3 pause 指令 …… (103)
5.5.4 keyboard 指令 …… (104)
5.5.5 break 指令 …… (104)
5.6 函数调用及变量传递 …… (104)
5.6.1 函数调用 …… (104)
5.6.2 参数传递 …… (106)
5.7 神经网络应用设计举例 …… (107)
5.7.1 带有偏差单元的递归神经网络 …… (107)
5.7.2 具有快速学习算法的补偿模糊神经网络 …… (118)
第六章 感知器 …… (132)
6.1 感知器的原理 …… (132)
6.1.1 感知器神经元模型 …… (132)

6.1.2　感知器神经元网络的网络结构 …… (133)
6.1.3　感知器神经网络的初始化 …… (134)
6.1.4　感知器神经网络的学习规则 …… (135)
6.1.5　感知器神经网络的训练 …… (135)
6.1.6　感知器的局限性 …… (135)
6.1.7　多层感知器 …… (136)
6.2　有关感知器的神经网络工具函数 …… (136)
6.2.1　MATLAB 中有关感知器的工具函数 …… (136)
6.2.2　工具函数详解 …… (137)
6.3　感知器网络设计实例 …… (157)
6.3.1　简单的分类问题 …… (157)
6.3.2　多个感知器神经元的分类问题 …… (160)
6.3.3　输入奇异样本对网络训练的影响 …… (162)
6.3.4　标准化感知器学习规则 …… (165)
6.3.5　线性不可分的输入向量 …… (166)
第七章　线性神经网络 …… (169)
7.1　线性神经网络原理 …… (169)
7.1.1　线性神经元模型 …… (169)
7.1.2　线性神经网络的模型 …… (170)
7.1.3　线性网络的初始化 …… (170)
7.1.4　线性网络的学习规则 …… (171)
7.1.5　线性网络的训练 …… (171)
7.2　有关线性网络的神经网络工具函数 …… (172)
7.2.1　MATLAB 中有关线性网络的工具函数 …… (172)
7.2.2　工具函数详解 …… (172)
7.3　线性网络设计及应用举例 …… (179)
7.3.1　线性网络设计实例 …… (179)
7.3.2　线性网络应用实例 …… (194)
第八章　BP 网络 …… (207)
8.1　BP 网络 …… (207)
8.1.1　BP 网络结构 …… (207)
8.1.2　BP 网络学习公式推导 …… (208)
8.2　MATLAB 神经网络工具箱中的 BP 网络 …… (212)
8.2.1　BP 网络中的神经元模型 …… (212)
8.2.2　BP 网络结构 …… (212)
8.2.3　MATLAB 中有关 BP 网络的重要函数 …… (213)
8.3　BP 网络的应用设计 …… (225)
8.3.1　BP 网络的训练过程 …… (225)
8.3.2　BP 网络的设计分析 …… (231)

8.4　BP 算法的改进及其设计实例 …… (232)
第九章　径向基函数网络 …… (244)
9.1　引言 …… (244)
9.2　径向基函数神经网络 …… (245)
9.3　径向基函数神经网络的工具箱函数 …… (246)
9.3.1　网络设计函数 …… (246)
9.3.2　权函数 …… (249)
9.3.3　网络输入函数 …… (250)
9.3.4　radbas（径向基）传递函数 …… (252)
9.3.5　mse 均方误差性能函数 …… (252)
9.3.6　变换函数 …… (253)
9.4　径向基函数网络的设计与应用 …… (254)
9.4.1　径向基函数网络的设计及在函数逼近上的应用 …… (254)
9.4.2　径向基函数网络与模糊理论的结合及应用 …… (260)
第十章　自组织竞争人工神经网络 …… (271)
10.1　两种联想学习规则 …… (271)
10.1.1　Instar 学习规则 …… (271)
10.1.2　Outstar 学习规则 …… (272)
10.2　基本竞争型人工神经网络 …… (273)
10.3　自组织特征映射神经网络 …… (274)
10.3.1　自组织特征映射网络的结构 …… (274)
10.3.2　自组织映射网络的学习及工作规则 …… (275)
10.4　自组织竞争人工神经网络工具箱函数 …… (278)
10.5　网络设计实例 …… (295)
10.5.1　竞争学习网络设计实例 …… (295)
10.5.2　一维空间自组织特征映射网络设计实例 …… (299)
第十一章　回归网络 …… (302)
11.1　引言 …… (302)
11.2　回归网络 …… (303)
11.2.1　Hopfield 网络 …… (304)
11.2.2　Elman 神经网络 …… (309)
11.3　回归网络的工具箱函数 …… (309)
11.3.1　Hopfield 网络的工具箱函数 …… (310)
11.3.2　Elman 网络的工具箱函数 …… (311)
11.4　应用举例 …… (312)
附录 A　MATLAB（5.1 版）神经网络工具箱函数 …… (329)
附录 B　MATLAB（5.3 版）神经网络工具箱函数 …… (331)

第一章 概 论

在现代高技术的发展过程中，神经网络理论应用得越来越广泛，它已是控制工程、信号处理等领域中不可缺少的工具，正日益受到科技人员的瞩目，并且随着一些挑战性的工程问题的解决，许多新的神经网络模型和算法正在不断地丰富着神经网络本身．然而，无论神经网络应用在哪个方面，当利用它对应用领域进行分析和设计的时候，都会涉及到大量有关数值计算的问题．这其中既包括一般的矩阵计算问题，如微分方法求解、优化问题等，也包括许多模式的正交化、最小二乘法处理和极大极小匹配等求解过程等等．尽管现代数值计算理论已经发展得很完善，多数计算问题都有高效的标准解法，但是利用计算机对神经网络应用系统进行仿真和辅助设计时，仍然是件很麻烦的事情．首先，编制程序是很繁杂的工作，需要不断找出错误（这几乎是不可避免），反复调试；其次，人机界面的设计很难做到令人满意的程度，计算结果缺乏强有力的图形输出支持；更重要的是，针对千变万化的应用对象，各类复杂的求解问题，编制一些特定的程序求解，耗费了大量的人力和物力．于是，许多科技人员希望选用现成的仿真软件工具，例如数值分析时用 Mathmatica，连续系统仿真时用 CSS 系列软件，以及通用仿真系统 GPSS 等．但这些软件往往只针对某一方面的问题有效，并且在人机接口、用户友好性等诸方面存在一定的缺陷．在微机飞速发展的今天，定位于 DOS 平台的软件已不再满足发展的需要．基于图形用户界面（GUI）的 MATLAB 软件包，为科技工程人员带来了巨大的便利，正越来越受到人们的青睐．本书结合神经网络的应用设计，分析 MATLAB 软件包语言和神经网络工具箱的特征、功能，并重点论述 MATLAB 神经网络仿真方法．

1.1 MATLAB 软件包的特征

1.1.1 MATLAB 语言（软件包）的发展过程

在 20 世纪 70 年代中期，Cleve Moler 博士和其同事在美国国家科学基金的资助下研究开发了 LINPACK 和 EISPACK 两个 FORTRAN 子程序库．LINPACK 是解线性方程的 FORTRAN 程序库，EISPACK 则是解特征值问题的程序库．

大约在 1980 年，Cleve Moler 博士在 New Mexico 大学讲线性代数课程时，想教学生使用 LINPACK 和 EISPACK 程序库，但又不希望学生在 FORTRAN 编程上花太多时间，出于这个目的，他开始用业余时间为学生编写使用 LINPACK 和 EISPACK 的接口程序．Cleve Moler 给这个接口程序取名为 MATLAB（MATrix LABoratory，即矩阵实验室）．

该软件出现后，一直在 New Mexico 大学作为教学辅助软件使用．后来，Cleve Moler 应邀去另一所大学讲学，在访问结束时他把 MATLAB 留在了那所大学的计算机上．从那以后，经过一年左右时间，MATLAB 开始受到欢迎，并成为应用数学界的术语．

随着 MATLAB 的影响不断扩大，斯坦福大学的工程师觉察到 MATLAB 的潜在应

用天地是工程领域，于是，1983 年初，他与 Moler，Steve Bangert 一起合作开发第二代专业版的 MATLAB. 1984 年 Moler 等专家觉得有必要组建一个软件开发公司（这个公司名 MathWorks），专门扩展并改进 MATLAB，并于 1993 年后相继推出了 MATLAB 4. X，MATLAB 5. X 等基于 Windows 系统的版本．

MATLAB 软件包研制的最初目的是为线性代数等课程提供一种方便可行的实验手段．由于该软件的使用极其容易，且提供丰富的矩阵处理功能，所以控制理论领域的研究人员很快就注意到了这个特点，并在它的基础上开发出了控制理论与 CAD 专门的应用程序集（又称工具箱）．于是，MATLAB 很快地在国际控制界流行起来．原先控制领域里的一些封闭式软件包则纷纷淘汰，这也是 MATLAB 最先应用的工程领域之一．据文献统计，目前 MATLAB 在学术界与工业界都是使用率最高的工具软件．

1. 1. 2 MATLAB 的功能

MATLAB 的核心是一个基于矩阵运算的快速解释程序．它以交互式接受用户输入的各项指令，输出计算结果．它提供了一个开放式的集成环境，用户可以运行系统提供的大量命令，包括数值计算、图形绘制等．具体来说，MATLAB 具有如下功能：

1）数据可视化功能；
2）强大的数值运算功能；
3）丰富的工具箱；
4）数学计算；
5）数字信号处理；
6）自动控制模拟；
7）动态分析；
8）数据处理 ；
9）2D/3D 的绘图功能；

并可以与 FORTRAN ，C，C＋＋做数据连接等等．

最新推出的 MATLAB 5. X 又增加了许多新的功能，其中包括：

1）增强了工具箱的功能；
2）增强了对多维矩阵的运算功能；
3）使用交互性更强的用户界面；
4）微分方程的新解法；
5）可以处理不同类型的向量与矩阵；
6）在 Simulink 中可以拥有自己的编辑器与调试器；
7）增加了许多功能函数及绘图功能．

1. 1. 3 MATLAB 的工具箱

MATLAB 的工具箱，为使用 MATLAB 的不同领域内的研究人员提供了捷径．迄今为止，已有 30 多种工具箱面世，内容涉及信号处理、自动控制、图像处理等领域．这些工具箱大致可分为两类：功能型工具箱和领域型工具箱．功能型工具箱主要用来扩充 MATLAB 的符号计算功能、图形建模仿真功能、文字处理功能以及与硬件实时交互功

能，能应用于多种学科．而领域型工具箱是专业性很强的，如控制工具箱、信号处理工具箱等．

具体地说，MATLAB 主要包括四大工具箱和近 30 个附属工具箱，有了这些工具箱，使 MATLAB 如虎添翼．四大工具箱除 MATLAB 本身，还有 Simulink，State Flow 和 Real-Time Workshop（RTW），四大工具箱的合作可以缩短整个产品生产的设计开发时间，减少设计所需要的成本．四大工具之间的关系如图 1.1 所示．

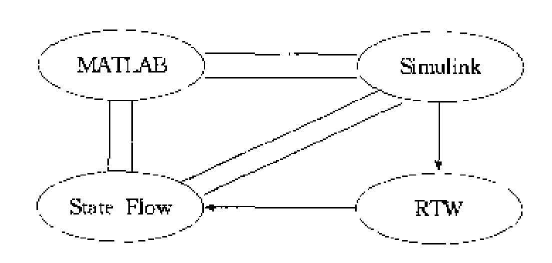


图1.1 四大工具之间的关系

应用于不同学科的附属工具箱有：通讯工具箱、控制系统工具箱、财政金融工具箱、频率域系统辨识工具箱、模糊逻辑工具箱、高阶谱分析工具箱、图像处理工具箱、线性矩阵不等式控制工具箱、模糊逻辑工具箱、分析与综合工具箱、神经网络工具箱、最优化工具箱、偏微分方程工具箱、鲁棒控制工具箱、信号处理工具箱、样条工具箱、统计工具箱、符号数学工具箱、系统辨识工具箱、小波工具箱、地图工具箱等．随着 MATLAB 版本的不断更新，MATLAB 工具箱会更加不断发展和完善．

综上所述，MATLAB 的功能和各种各样的工具箱，使它具有如下特点：

1）开放式的体系结构；

2）先进的界面技术；

3）丰富的技术支持；

4）集成了许多领域专家的智慧．

1.2 MATLAB 的系统需求和安装

1.2.1 MATLAB 的系统需求

（1）硬件要求：

1）中央处理器（CPU）如果是 INTEL486 需加上浮点运算器 487，486DX 也必须内含浮点运算器．建议采用 586.

2）内存要求最小为 8MB，建议采用 16MB 以上的内存配置．

3）光驱（CD-ROM）．

4）VGA 显示卡和显示器，屏幕最少能显示 256 色，建议配备 3D 图形加速卡．

5）硬盘可用空间最少为 100MB.

6）鼠标（Mouse）虽非必需，但为了能在 Windows 下工作轻松而迅速，建议使用．

7）有少量指令需要声卡．

（2）软件要求：

1）操作系统为 Windows95/98，Windows NT.

2）当使用时 Notebook 时，还需要 MS-Word6.0 以上的中英文版.

1.2.2 MATLAB 的安装

MATLAB5.3 有三种版本，分别是 Windows95 版，Windows NT 和 Macintosh 版，这里主要介绍 Windows95 版，因为读者所使用的大部分是 Windows98 系统.

MATLAB5.3 比较容易安装，不需要另外设定其他参数、同安装其他软件一样、具体安装步骤如下：

步骤 1

在 Windows95/98 操作平台中将 MATLAB 的原版光盘放入 CD-ROM 中，系统将会自动安装程序. 如果没有执行，双击 setup.exe 就可以开始安装 MATLAB 了.

步骤 2

接下来在安装界面中会看到 MATLAB 的欢迎对话框，如图 1.2 所示.

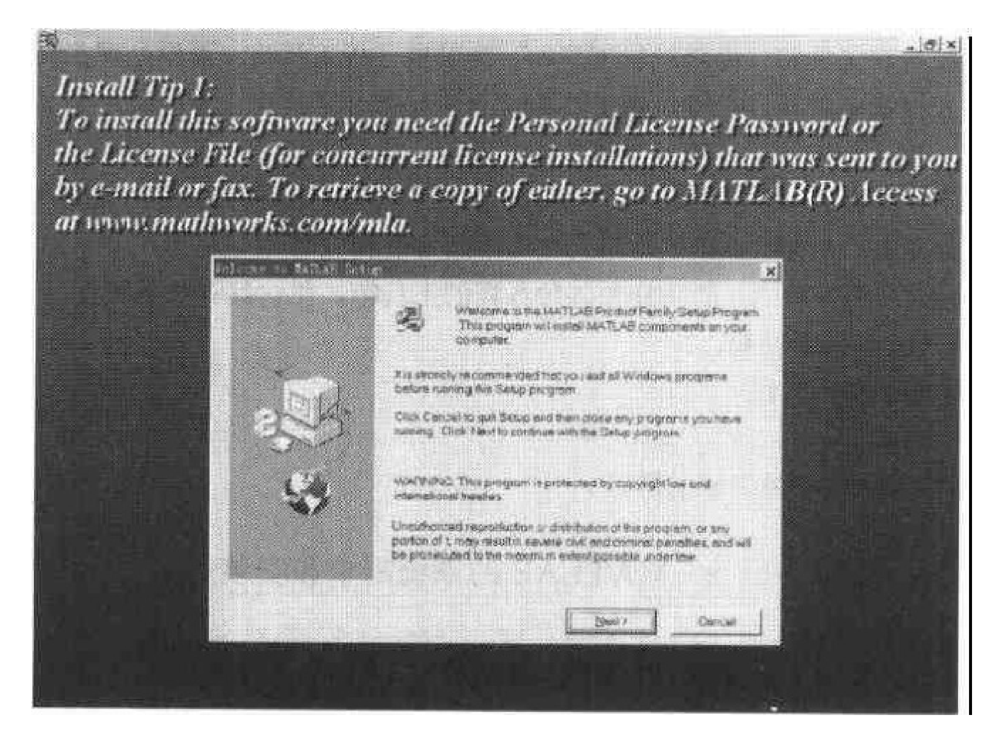


图 1.2　欢迎对话框

单击 Next，开始下一步.

步骤 3

屏幕上会出现如图 1.3 所示的界面，其内容为软件的使用许可和相关信息. 如果接受就单击 Yes，否则单击 No. 单击 Yes 进一步安装.

步骤 4

接着会出现如图 1.4 所示的信息窗口，在这个窗口中，输入您的姓名、公司名称和序列号，然后选择版本号. 单击 Next 进一步安装.

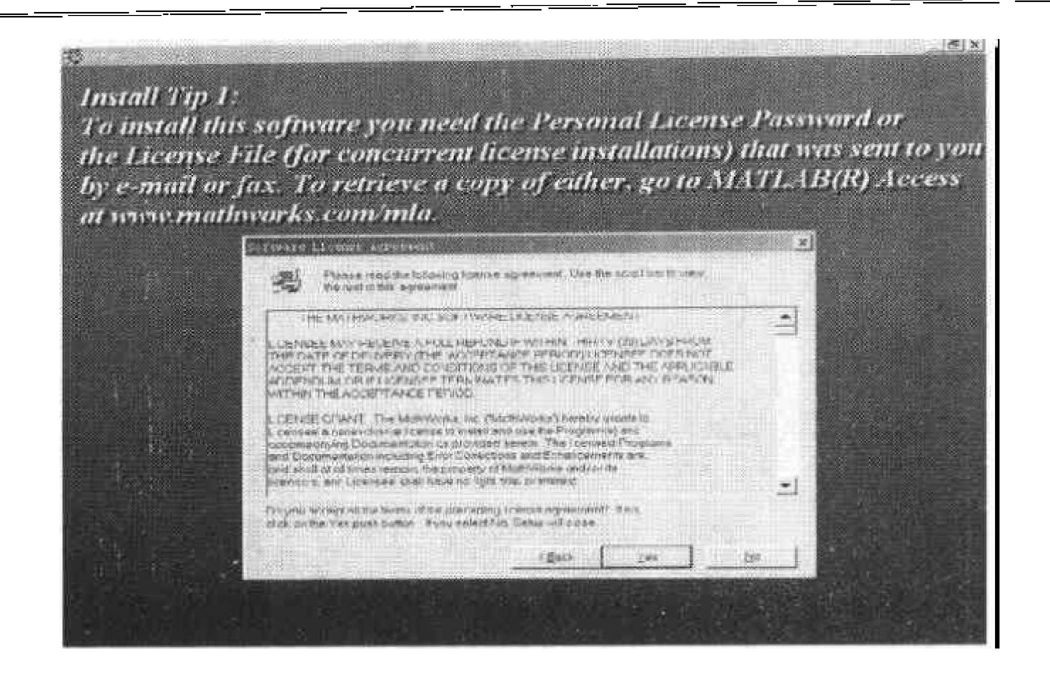


图 1.3 软件的使用许可

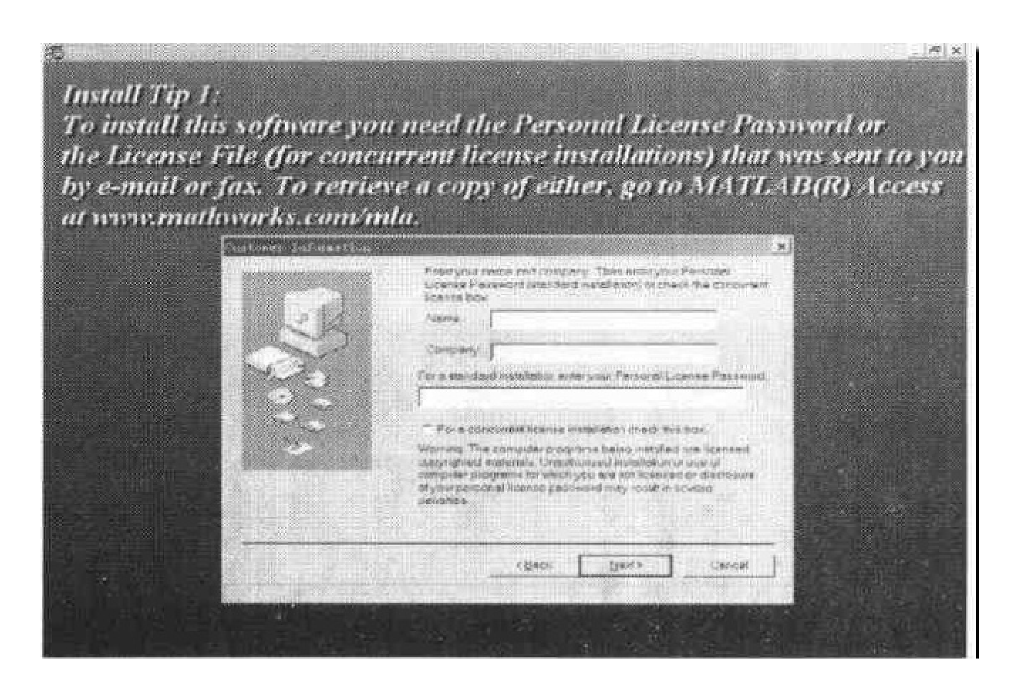


图 1.4 信息窗口

步骤 5

在出现的安装组件对话框中选择你所需要安装的工具箱，也可以选择不要安装，而在以后用到时再安装．缺省安装路径为 C：\MATLAB，如果想更改安装路径，选择 Browse 按钮，如图 1.5 所示．如果选择安装辅助说明文件的话，辅助说明文件将全部安装在 MATLAB Help Desk 目录下；应该全部安装或者干脆一点也不要安装，但是绝对不要只将其中一部分安装在硬盘上．单击 Next 继续安装．

注意：硬盘可用空间（space available）必须比 MATLAB 所需安装空间（space required）大．

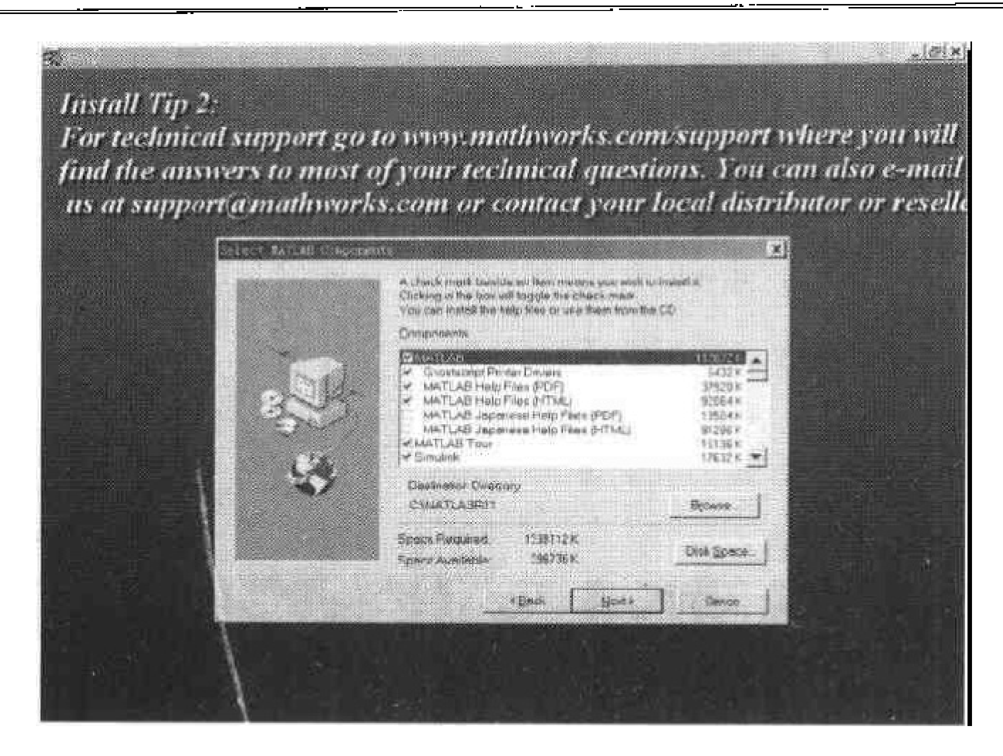


图 1.5 选择需要安装的工具箱

最好不要用MATLAB 高版本覆盖以前安装的低版本．应该选择新的安装目录．当用新的MATLAB 版本覆盖以前安装的低版本时，最好在开始安装前备份你所编写的所有 m 文件．

步骤 6

如图 1.6 所示，正在安装．

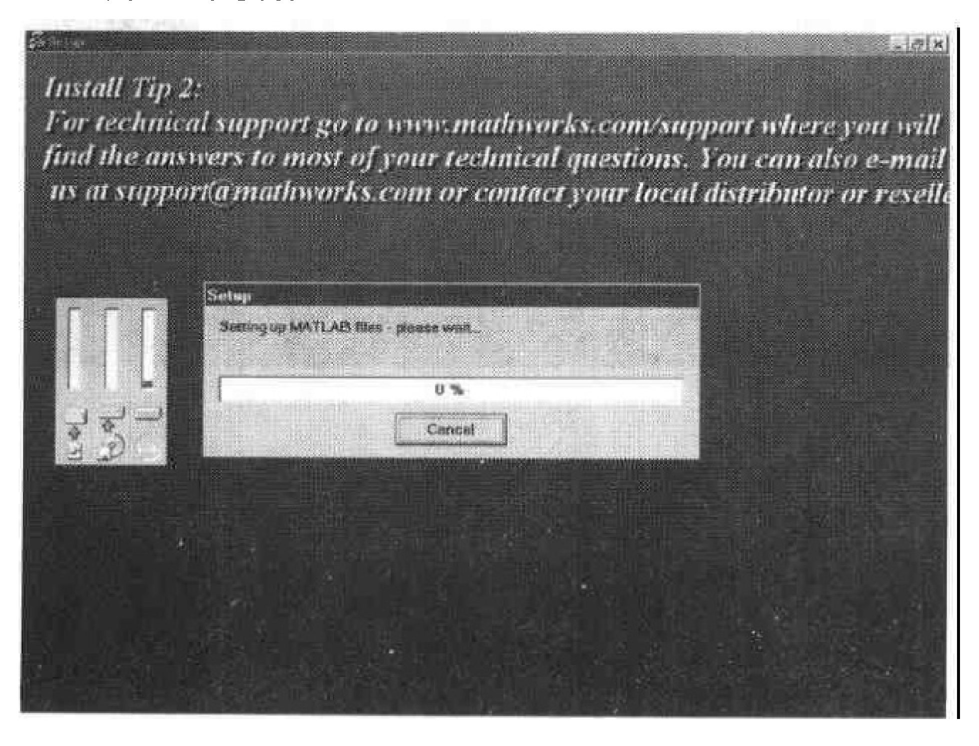


图 1.6 正在安装

步骤 7

成功地安装 MATLAB 后，就会看到完成画面，如图 1.7 所示.

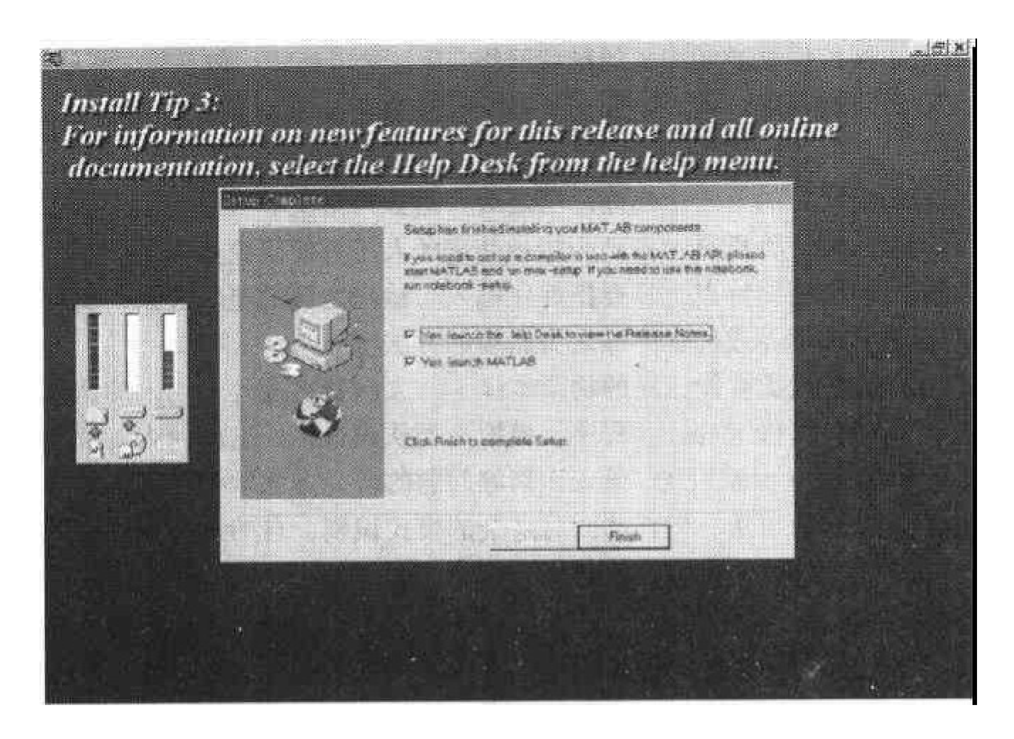


图 1.7 安装成功

MATLAB 安装成功后，进入 help desk 查看在线帮助文件（help desk 以 .pdf 和 .html 两种格式提供 MATLAB 的帮助文件）. 函数方程的相关文件都在 help desk 中.

1.3 神经网络发展和应用

1.3.1 人工神经网络发展的历史回顾

神经网络的研究已有较长的历史，一般可以用两次热潮来划分其历史时期.

(1) 第一次热潮（1943 年至 1969 年）

早在 1943 年，心理学家 W. McCulloch 和数学家 W. Pitts 合作提出的兴奋与抑制型神经元模型和 Hebb 提出的神经元连接强度的修改规则，开创了神经科学理论研究的时代.

1948 年，Wiener 在其控制论的著作中提出了伺服机反馈自稳定系统概念.

1958 年，F. Rosenblatt 首次引进了模拟人脑感知和学习能力的感知器概念，引起了人们的注意.

1961 年，Caianiello 发表了关于神经网络数学的理论著作，提出了神经元网络方程，将神经元作为双态器件，对其机能的动力过程用布尔代数加以模拟，进而分析和研究了细胞有限自动机的理论模型.

1962 年，B. Widrow 提出的自适应线性元件（adaline），具有自适应学习功能，在信号处理、模式识别等方面受到普遍重视和应用.

在上述的数学模拟理论发展的同时，还出现了利用电子器件进行电子模拟的跑迷津的金属老鼠、条件概率机的龟模型，以及会下棋的机器等，形成了脑模拟研究的第一个热潮．在这期间，神经网络大都是单层线性网络．

感知器是由阀值性神经元组成的层状网络，具有学习功能．但是，它也有局限性，如不能产生复杂的逻辑函数，同时，又由于数字计算机正处于全盛时期并在人工智能领域取得显著成就，从而使得人工神经网络的研究处于低潮．

人工神经网络研究的第一个热潮冷落之后，对如何解决非线性分割问题很快地有了明确的认识，但此时计算机科学界已为阔步前进的人工智能研究热潮所笼罩，似乎不需要考虑脑功能的网络结构特点，只要把握住输入与输出之间的逻辑关系，就可以实现人工智能，这就是心理学界或科学界称之为黑箱的方法学原则．

人工智能研究把人类智能活动的物质本体——大脑置之度外．用人的认识规律编写计算机软件，再由计算机运行这些程序，模拟人类认识过程，这就使人工智能研究在不足 30 年的历史中，便在专家系统、语音识别等问题的研究方面取得了引人瞩目的辉煌成果，成为计算机科学的明珠．但在其面对复杂的模式识别、自然语言理解和机器人自适应控制等方面，常常处于无能为力的窘境．

进入 80 年代后，传统的数字计算机在模拟视听觉的人工智能方面遇到了物理上不可逾越的极限．与此同时，物理学家 Hopfield 提出了 HNN 模型，引人了能量函数的概念，给出了网络稳定性的判据，神经网络的热潮再次掀起．

（2）第二次热潮

1982 年，在美国国家科学院的刊物上发表了著名的“Hopfield”模型理论，这是一个非线性动力系统的理论模型．它引起了各国学者的关注，并力图将这一数学模型进行电子学或光学的硬件实现，这就形成了一大批人工神经网络的研究队伍．当代 80％的各国研究者都是追随 Hopfield 模型而迈人这一学术领域中．这一模型对人工神经网络信息存贮和提取功能进行了非线性数学概括，提出了动力方程和学习方程，使人工神经网络的构造和学习有了理论指导．在 Hopfield 提出之后，许多学者力图将这一模型扩展，使之更接近人脑的功能特性．1983 年，即 Hopfield 模型提出的第二年，年轻学者 Sejnowski 与其合作者 Hinton 提出了大规模并行网络（massively parallell）学习机，并明确提出隐单元(hidden unit)的概念．这种学习机后来称之为 Boltzmann 机．他们应用多层神经网络并行分布地改变各神经元间的连接权，克服了以往神经网络的局限性．此外，Sejnowski 还运用这些原理构造了著名的 NETtalk 程序系统．

K. Fukushima 在 Rosenblatt 的知觉器网络基础上增加了隐层，构成了多层认识器，实现了可塑的反馈联系和更为普遍的前馈联系．这种认知器通过抑制性反馈和兴奋性前馈作用，即使外部刺激停止以后，也可以继续实现自组织自学习过程．还有 J. Anderson 提出了 BSB 模型，S. Grossberson 提出了自适应共振理论，T. Kohonen 提出了自组织映射网络，D. Willshaw 等提出了联想记忆网络，这些努力为神经网络的后期发展奠定了牢固的基础．目前在研究方法上已形成多个流派、包括多层网络 BP 算法，Hopfield 网络模型，自适应共振理论（ART），自组织特征映射理论等．1987 年 IEEE 在 San Diego 召开了甚大规模的神经网络国际学术会议，国际神经网络学会也随之诞生，随后不久学会的杂志《神经网络》和 IEEE 的神经网络杂志相继问世．

迄今为止的神经网络研究，大体上可分为三个大的方向：

1）探求人脑神经系统的生物结构和机制，这实际上是神经网络理论的初衷；

2）用微电子学或光学器件形成特殊功能网络，这主要是新一代计算机制造领域所关注的问题；

3）将神经网络理论作为一种解决某些问题的手段和方法，这些问题在利用传统方法时或者无法解决，或者在具体处理技术上尚存困难．

1.3.2 神经网络的应用

神经网络理论的应用取得了令人瞩目的进展，特别是在人工智能、自动控制、计算机科学、信息处理、机器人、模式识别、CAD/CAM 等方面都有重大的应用实例．下面列出一些主要的应用领域．

1）模式识别和图像处理

印刷体和手写体字符识别、语音识别、签字识别、指纹识别、人脸识别、人体病理分析、目标检测与识别、图像压缩和图像复原等．

2）控制和优化

化工过程控制、机器人运动控制、家电控制、半导体生产中掺杂控制、石油精炼优化控制和超大规模集成电路布线设计等．

3）预报和智能信息管理

股票市场预测、地震预报、有价证卷管理、借贷风险分析、IC 卡管理和交通管理．

4）通讯

自适应均衡、回波抵消、路由选择和 ATM 网络中的呼叫接纳识别及控制等．

5）空间科学

空间交会对接控制、导航信息智能管理、飞行器制导和飞行程序优化管理等．

1.3.3 神经网络的学习方法

1．学习方式

通过向环境学习获取知识并改进自身性能是神经网络的一个重要特点．在一般情况下，性能的改善是按某种预定的度量通过调节自身参数（如权值）逐步达到的．学习方式有三种．

（1）监督学习（有教师学习）

如图 1.8 所示，这种学习方式需要外界存在一个"教师"，他可对给定一组输入提供应有的输出结果，这组已知的输入-输出数据称为训练样本集，学习系统（神经网络）可根据已知输出与实际输出之间的差值（误差信号）来调节系统参数．

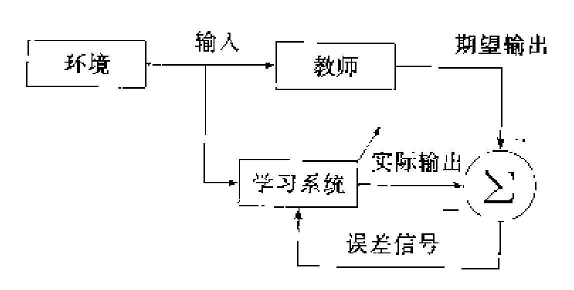


图1.8 有教师指导的学习框图

（2）非监督学习（无教师学习）

如图 1.9 所示，非监督学习时不存在外部教师，学习系统完全按照环境提供数据的

某些统计规律来调节自身参数或结构（这是一种自组织过程），以表示出外部输入的某种固有特性（如聚类或某种统计上的分布特征）.

（3）再励学习（强化学习）

如图 1.10 所示，这种学习介于上述两种情况之间，外部环境对系统输出结果只给出评价信息（奖或惩）而不是给出正确答案．学习系统通过强化那些受奖的动作来改善自身的性能．



图1.9　无教师指导的学习框图　　图1.10　再励学习框图

2. 学习算法（学习规则）

（1）误差纠正学习

令 $y_k(n)$ 为输入 $x_k(n)$ 时，神经元在 n 时刻的实际输出，$d_k(n)$ 表示应有的输出（可由训练样本给出），则误差信号可表示为

$$e_k(n)=d_k(n)-y_k(n)$$

误差纠正学习的最终目的是使某一基于 $e_k(n)$ 的目标函数达到最小，以使网络中每一输出单元的实际输出在某种统计意义上逼近应有输出．一旦选定了目标函数形式，误差纠正学习就变成了一个典型的最优化问题．最常用的目标函数是均方误差判据，定义为误差平方和的均值

$$J=E\left[\frac{1}{2}\sum_k e_k^2(n)\right]$$

其中 E 为求期望算子，上式的前提是被学习的过程是宽平稳的，具体方法可用最优梯度下降法．直接用 J 作为目标函数时需要知道整个过程的统计特性，为解决这一问题，通常用 J 在时刻 n 的瞬时值 $\xi(n)$ 代替，即

$$\xi(n)=\frac{1}{2}\sum_k e_k^2(n)$$

问题变为求 $\xi(n)$ 对权值 W 的极小值，根据梯度下降法可得

$$\Delta w_{kj}=\eta e_k(n)x_j(n)$$

其中 η 为学习步长，这就是通常所说的误差纠正学习规则（或称 delta 学习规则）．在自适应滤波器理论中，对这种学习的收敛性及其统计特性有较深入的分析．

（2）Hebb 学习

由神经心理学家 Hebb 提出的学习规则可归纳为“当某一突触（连接）两端的神经元同步激活（同为激活或同为抑制）时，该连接的强度应增强，反之应减弱”．用数学方式可描述为

$$\Delta w_{kj}=F(y_k(n)x_j(n))$$

式中 $y_k(n)$，$x_j(n)$ 分别为 w_{kj} 两端神经元的状态，其中最常用的一种情况是

$$\Delta w_{kj}=\eta y_k(n)x_j(n)$$

由于 Δw_{kj} 与 $y_k(n)$、$x_j(n)$ 有关，有时称为相关学习规则．

(3) 竞争 (competitive) 学习

顾名思义，在竞争学习时，网络各输出单元互相竞争，最后达到只有一个最强者激活，最常见的一种情况是输出神经元之间有侧向抑制性连接（如图1.11所示），这样原来输出单元中如有某一单元较强，则它将获胜并抑制其他单元，最后只有此强者处于激活状态．最常用的竞争学习规则可写为

$$\Delta w_{ji}=\eta(x_i-w_{ji}), \text{若神经元 } j \text{ 竞争获胜}$$
$$\Delta w_{ji}=0, \qquad \text{若神经元 } j \text{ 竞争失败}$$

3．学习与自适应

当学习系统所处环境平稳时（统计特性不随时间变化），从理论上讲通过监督学习可以学到环境的统计特性，这些统计特性可被学习系统（神经网络）作为经验记住．如果环境是非平稳的（统计特性随时间变化），通常的监督学习没有能力跟踪这种变化，为解决此问题，需要网络有一定的自适应能力，此时对每一不同输入都作为一个新的例子来对待．其工作过程如图1.12所示，此时模型（即神经网络）被当作一个预测器，基于前一时刻输入 $x(n-1)$ 和模型在 $(n-1)$ 时刻的参数，它估计 n 时刻的输出 $\hat{x}(n)$，$\hat{x}(n)$ 与实际值 $x(n)$（作为应有的正确答案）比较，其差值称为"新息"，如新息 $e(n)=0$，则不修正模型参数，否则应修正模型参数以便跟踪环境变化．

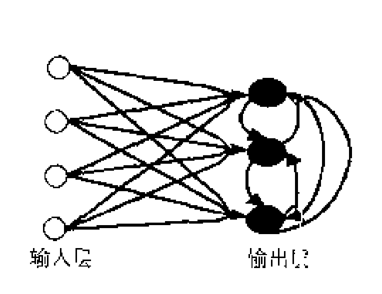


图1.11 具有侧向抑制性连接的竞争学习网络

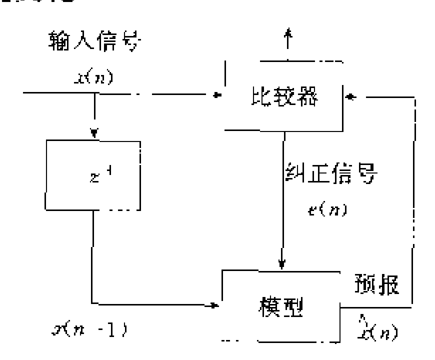


图1.12 自适应系统框图

1.4 面向MATLAB工具箱的神经网络设计概述

1.4.1 MATLAB神经网络工具箱

神经网络工具箱是在MATLAB环境下所开发出来的许多工具箱之一，它是以人工神经网络理论为基础，用MATLAB语言构造出典型神经网络的激活函数，如S型、线性、竞争层、饱和线性等激活函数，使设计者对所选定网络输出的计算变成对激活函数的调用．另外，根据各种典型的修正网络权值的规则，加上网络的训练过程，用MATLAB编写出各种网络设计与训练的子程序，网络的设计者则可以根据自己的需要去调用工具箱中有关神经网络的设计训练程序，使自己能够从繁琐的编程中解脱出来，集中精力去思考问题和解决问题，从而提高效率和解题质量．

目前 MATLAB 最新版本几乎完整地概括了神经网络的基本成果，所涉及到的网络模型有：

1）感知器；

2）线性网络；

3）BP 网络；

4）径向基函数网络；

5）自组织网络；

6）回归网络．

对于各种网络模型，神经网络工具箱集成了多种学习算法，为用户提供了极大的方便．此外，神经网络工具箱中还给出了大量的示例程序，为用户轻松地使用工具箱提供了生动实用的范例．

MATLAB 的其他工具箱（如模糊逻辑工具箱、样条工具箱等）也为我们在神经网络工具箱的基础上开发研究模糊与神经网络的结合、神经网络的样条算法等问题提供了辅助手段．

1.4.2 神经网络技术的选取

在决定采用神经网络技术之前，应首先考虑是否有必要采用神经网络来解决问题．一般地，神经网络与经典计算方法相比并非优越．只有当常规方法无法解决或效果不佳时神经网络才能显示出其优越性．尤其是当对问题的机理等规律不甚了解，或不能用数学模型表示的系统，如故障诊断、特征的提取和预测、非线性系统的自适应控制等问题，神经网络往往是最有利的工具．另一方面，神经网络对处理大量原始数据而不能用规则或公式描述的问题，表现出极大的灵活性和自适应性．

就目前看，MATLAB 神经网络工具箱所给出的神经网络模型主要适用的应用范围有：函数和模型逼近、信号处理和预测、分群、自适应控制、故障诊断推理等．

1.4.3 运用工具箱设计网络的原则和过程

在实际应用中，面对一个具体的应用问题时，首先要分析用神经网络方法求解问题的性质，然后根据问题特点，确定网络模型．最后，通过网络仿真分析，确定网络是否适合、是否需要修改．其具体过程如下：

（1）确定信息表达方式

将领域问题及其相应的领域知识转化为网络所能表达并能处理的形式，即将领域问题提炼成适合网络求解所能接受的某种数据形式．尽管应用问题会多样化，但数据形式大致有以下几类：

1）已知数据样本；

2）已知一些相互关系不明的数据样本；

3）输入/输出模式为连续量、离散量；

4）希望把所有输入数据按模式分类、希望识别具有平移、旋转、伸缩等变化的模式．

（2）网络模型选择

这主要包括确定激活函数、联接方式、各神经元的相互作用等．另外还可在典型网

络模型的基础上，结合具体应用问题特点，对原网络模型进行变型、扩充，也可采用多种网络模型的组合等．

（3）网络参数选择

确定输入、输出神经元的数目、多层网的层数和隐层神经元的数目，还有一些递归神经元等问题．

（4）学习训练算法选择

确定网络学习训练时的学习规则及改进学习规则．在训练时，还要结合具体的算法，考虑初始化问题．

（5）系统仿真的性能对比实验

将应用神经网络解决的领域问题与其他采用不同方法的仿真系统的效果进行比较．

总而言之，MATLAB 软件包充分利用了现代计算机技术所提供的软硬件资源和先进思想，是一个非常先进而且方便的最流行的软件．在国内外，它已逐渐成为大学生必须掌握的一种工具软件．而 MATLAB 环境下的神经网络工具将使设计研究者如虎添翼，有时间思考和研究出功能更强、更有效的神经网络和神经网络应用成果．

1.5 本书的基本结构和内容概要

第一章 概论

主要介绍MATLAB 发展历史、总体功能概貌，系统硬件支持条件和使用环境；另外，介绍神经网络工具箱及理论发展背景知识等，最后给出本书的基本结构和内容．

第二章 MATLAB 数值计算功能

本章叙述数值矩阵的各种运算、功能函数的使用方法、矩阵分解、线性方程的求解、多项式运算、数据分析和数值信号处理等 MATLAB 数值计算功能．

第三章 MATLAB 符号处理

本章叙述符号矩阵的运算、分解和微积分及微分方程的求解等，此外，还要叙述字符串的逻辑操作等．

第四章 绘图

介绍 MATLAB 的 2D、3D 绘图方法、图形的控制与表现、色彩控制、动态图形及图形的输出和打印，图形句柄的操作等功能和使用技巧，其中还包括一些实验例子．

第五章 MATLAB 的程序设计

本章叙述 m 文件功能、形式、程序结构、数据结构、变量定义和程序流控制，函数调用，数据的输入输出；本章还给出一些 MATLAB 语言设计的神经网络模型和训练的例子（如 BP 网、模糊神经网络等）．

第六章 感知器

本章首先描述感知器的网络结构、学习规则和训练过程，然后介绍MATLAB 工具箱与感知器相关的函数，最后给出感知器的设计和应用实例．

第七章 线性神经网络

描述线性神经网络的网络结构、学习规则和训练过程，然后介绍MATLAB 工具箱与线性神经网络相关的函数，最后给出线性神经网络的设计及在预测和系统辨识应用中的

几个实例．

第八章　BP 网络

首先描述 BP 的网络结构、学习规则和训练过程，分析改进 BP 网的算法，然后介绍 MATLAB 工具箱与 BP 网相关的函数，还要介绍结合 BP 网络的工具箱，探讨一些如改进 BP 网的设计等扩展性的应用问题，最后给出 BP 网络的设计和系统建模中的应用实例．

第九章　径向基函数网络

本章首先描述径向基函数网络结构、理论和训练过程，然后介绍 MATLAB 工具箱与径向基函数网络相关的函数及各种径向基网络的设计与仿真分析，最后给出径向基函数网络的设计和在状态观测器设计中的应用．

第十章　自组织竞争网络

叙述几种联想学习规则，自组织网络结构和它们的学习训练过程，然后介绍 MATLAB 工具箱与自组织竞争网络相关的函数，最后给出这些网络的设计和在特征映射方面的几个实例．

第十一章　回归网络

叙述回归网络的网络结构(Hopfield，Elman)、学习规则和训练过程，然后介绍 MATLAB 工具箱与回归网络相关的函数，最后给出回归网络的设计和在模式分类应用中的几个实例．

附录

MATLAB 主要函数指令及神经网络工具箱函数一览表．

第二章　MATLAB 数值计算功能

数学计算有数值计算和符号计算之分，它们的根本区别主要体现在：前者的表达式、矩阵变量中不允许包含未定义的自由变量而后者允许．

另外，还有几点需要说明：

1) MATLAB 是以矩阵为基本运算"单元"．这给编写程序带来很大方便．用户可通过 MATLAB 来处理线性代数的运算，快速准确地完成一些复杂、易出错而且费时的运算工作．

2) 在 MATLAB 中，不管是数值矩阵还是符号矩阵都不必事先定义维数大小、MATLAB 会根据用户所输入的矩阵结构自动配置、并在此后的运算中按正确的数学法则自动地调整矩阵的维数．

2.1　矩阵与数组运算

2.1.1　矩阵的建立

通常矩阵与数组的意义相同，都是指含有 M 行与 N 列数字的矩形结构．矩阵中的元素可以是实数或者复数．要用 MATLAB 做矩阵运算，首先要将矩阵输入到 MATLAB 中．下面给出两种创建数值矩阵的直接输入法．

例 2.1　在 MATLAB 环境下，用下面三条指令创建数值阵 C，如图 2.1 所示．

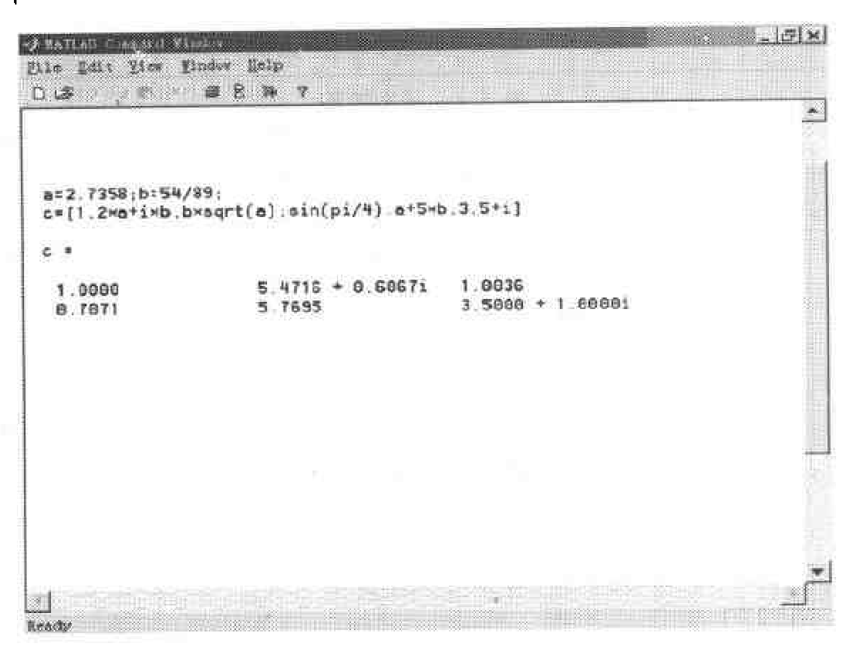


图 2.1　范例 2.1

说明：1）分号"；"的三个作用：

a．在"[ ]"方括号内时，它是矩阵行间的分隔符．

b. 它可用作指令与指令间的分隔符.

c. 当它放在赋值指令后时，该指令执行后的赋值结果将不显示在屏幕上.

2）指令中的 "pi" 和 "i" 都是MATLAB的预定义变量（predefined variable）名：

a. "pi" 代表圆周率 π.

b. "i" 代表虚数单位，$i=\sqrt{-1}$.

例 2.2 复数矩阵的另一种输入方式，如图 2.2 所示.

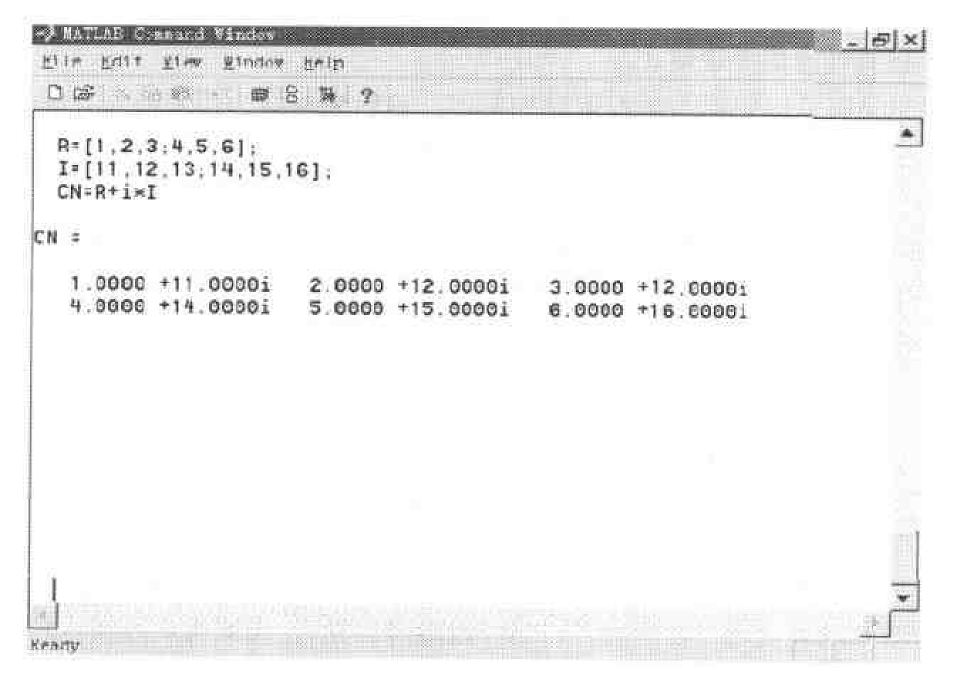


图 2.2 范例 2.2

除了可以运用上述方法创建矩阵之外，也可以使用下面（表 2.1～2.5）的函数来生成.

表 2.1 基本矩阵

函　数	功 能 描 述
zeros	产生零矩阵
ones	产生全部元素均为 1 的矩阵
eye	产生单位矩阵
repmat	产生元素区块重复矩阵
rand	产生均匀分布随机数矩阵
randn	产生正态分布随机数矩阵
linspace	产生线性等间距的列向量
logspace	产生对数等间距的列向量
meshgrid	产生用于 3D-plots 的 X 和 Y 数组

表 2.2　基本矩阵信息

函　　数	功 能 描 述
size	矩阵大小
length	向量长度
ndims	数组的维数
disp	显示数组或文字
isempty	检测空矩阵
isequal	检测矩阵是否相等
isnumeric	检测数值数组
islogical	检测逻辑数组
logical	转换数值数组为逻辑数组

表 2.3　矩阵运算操作

函　　数	功 能 描 述
reshape	更改矩阵大小
diag	对数数组或矩阵
tril	取出矩阵的下三角部分
triu	取出矩阵的上三角部分
fliplr	将矩阵左右对调
flipud	将矩阵上下对调
flipdim	将矩阵沿特定方向对调
rot90	将矩阵旋转 90°
find	找出非零元素的索引
end	最后元素的索引
sub2ind	将下标转化为线性索引
ind2sub	将线性索引转化为下标

表 2.4　特殊变量和常数

函　　数	功 能 描 述
ans	最近一次执行的结果，如果不指定执行的输出，MATLAB 会将结果保存在 ans
eps	浮点数的精确度，换句话说，就是执行 MATLAB 运算时的准确度
realmax	用户电脑能表示的最大的正浮点数值
realmin	用户电脑能表示的最小的正浮点数值
pi	圆周率 π
i、j	复数
inf	无限大的值，即一个数除以零的结果
NaN	不是一个数值，例如 0/0 或 inf/inf 都会产生这个结果
isnan	元素为非数字时为真
isinf	元素为正（负）无穷时为真
isfinite	元素有限时为真
flops	计算到当前为止系统执行基本浮点运算的次数
why	提供几乎任何问题的简捷回答

表 2.5　特殊矩阵

函　数	功 能 描 述
compan	伴随矩阵
gallery	Higham 检测矩阵
hadamard	Hadamard 矩阵
hankel	Hankel 矩阵
hilb	Hilbert 矩阵
invhilb	Hilbert 逆矩阵
magic	幻方
pascal	Pascal 矩阵
rosser	经典的对称特征值检验问题
toeplitz	Topeplitz 矩阵
vander	Vandermonde 矩阵
wilkinson	Wilkinson 特征值检验矩阵

2.1.2　矩阵和数组运算指令对照汇总

表 2.6 扼要地罗列两种运算指令形式和实质的异同点.

说明:

1) 数组四则运算、乘法、转置运算符中的小黑点绝对不能遗漏，否则将不按数组运算规则进行计算.

2) 不管执行什么数组运算，所得计算结果数组总是与参与运算的数组维数同维.

3) 要特别注意两种运算在乘、除和乘方方面的本质区别.

对于上述指令，可以从下面示例中体会其用法（由于加、减、转置等运算非常简单，指令也容易理解，所以下面不给算例）.

表 2.6　两种运算指令形式和实质内涵的异同表

矩阵运算指令	指 令 含 义	数组运算指令	指 令 含 义
A'	矩阵共轭转置	A.'	矩阵元素的共轭转置,相当于 conj(A')
A+B	矩阵相加	A.+B	对应元素相加(与矩阵相加等效)
A−B	矩阵相减	A.−B	对应元素相减(与矩阵相减等效)
s+B	标量与矩阵相加(MATLAB 约定的特殊运算)		
s−B,B−s	标量与矩阵相减(MATLAB 约定的特殊运算)		
A*B	内维相同矩阵的乘	A.*B	同维数组对应元素相乘
s*A	A 的每个元素乘 s	s.*A	A 的每个元素乘 s(标量和矩阵相乘结果与此同)
A/B	A 右除 B	A./B	A 的元素被 B 的对应元素除
B\A	A 左除 B	A.\B	A 的元素被 B 的对应元素除
inv(B)	B 阵的逆		
		s./B,B.\s	s 分别被 B 的元素除
A^n	A 阵的 n 次幂	A.^n	A 的每个元素自乘 n 次
A^p	A 阵的非整数乘方	A.^p	A 的每元素分别求非整数乘方
p^A	标量的矩阵乘方	p.^A	以 p 为底,分别以 A 的元素为指数求幂

例 2.3　矩阵运算示例（计算 CA^2B），如图 2.3 所示.

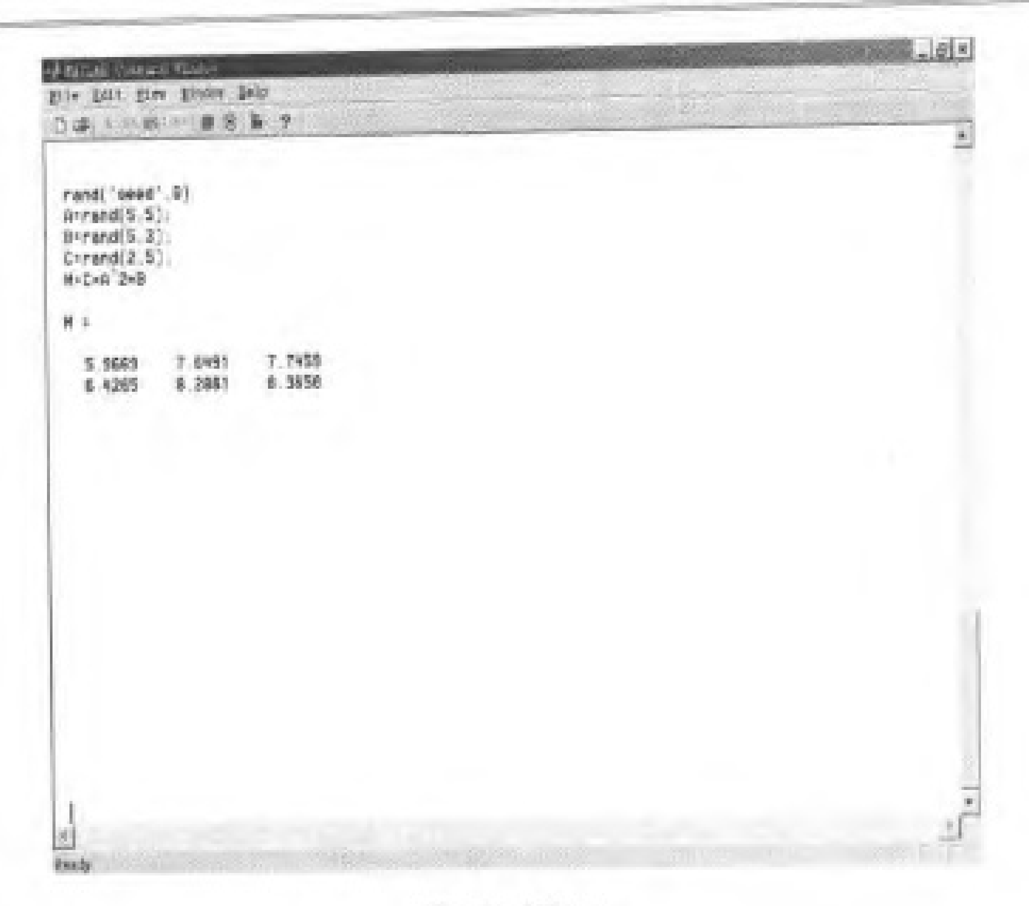


图 2.3　范例 2.3

例 2.4　矩阵乘和数组乘的不同，如图 2.4 所示。

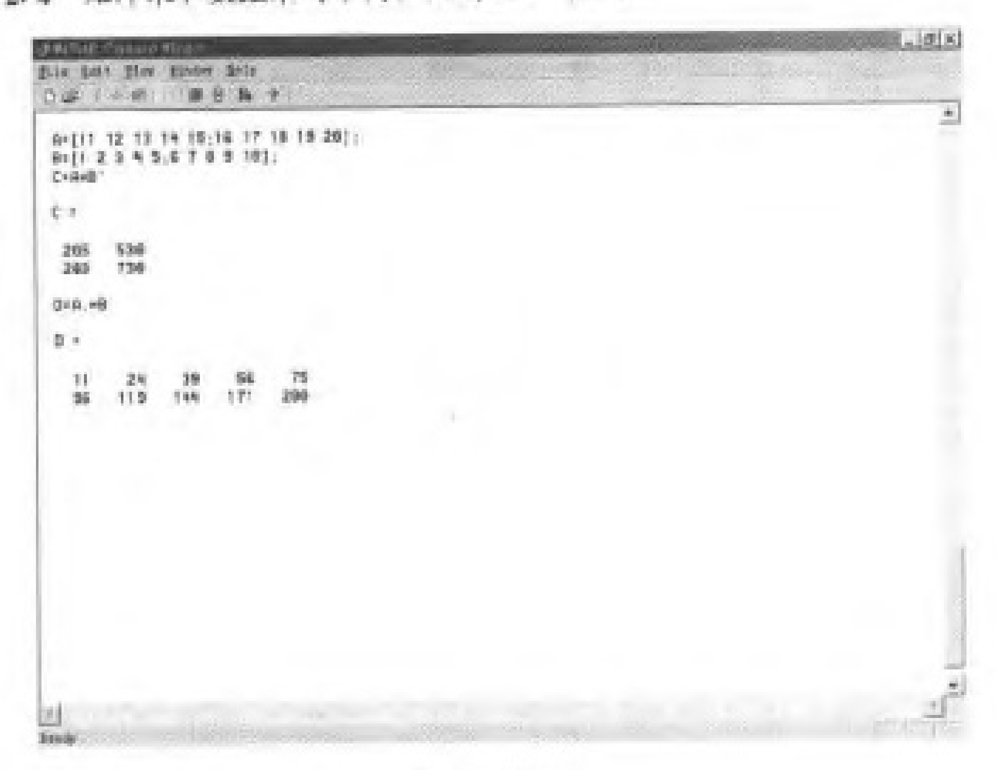


图 2.4　范例 2.4

例 2.5　计算 x=A\b，如图 2.5 所示．

```
A=fix(15*rand(3,3))

A =

     7    13    14
     5     7     8
     4     6    11

b=fix(15*rand(3,1))

b =

    11
    12
     1

x=A\b

x =

   -0.2770
    1.9122
   -0.8514

C=A*x

C =

    11
    12
     1
```

图 2.5　范例 2.5

例 2.6　求矩阵的非整数乘方，如图 2.6 所示．

```
A=[1 2 3;4 5 6;7 8 9]

A =

     1     2     3
     4     5     6
     7     8     9

Ap=A^0.3

Ap =

  0.6962 + 0.6032i   0.4358 + 0.1636i   0.1755 - 0.2755i
  0.6325 + 0.0666i   0.7309 + 0.0181i   0.8292 - 0.0305i
  0.5688 - 0.4700i   1.0255 - 0.1275i   1.4820 + 0.2150i
```

图 2.6　范例 2.6



例 2.7 数组的标量乘方，如图 2.7 所示.

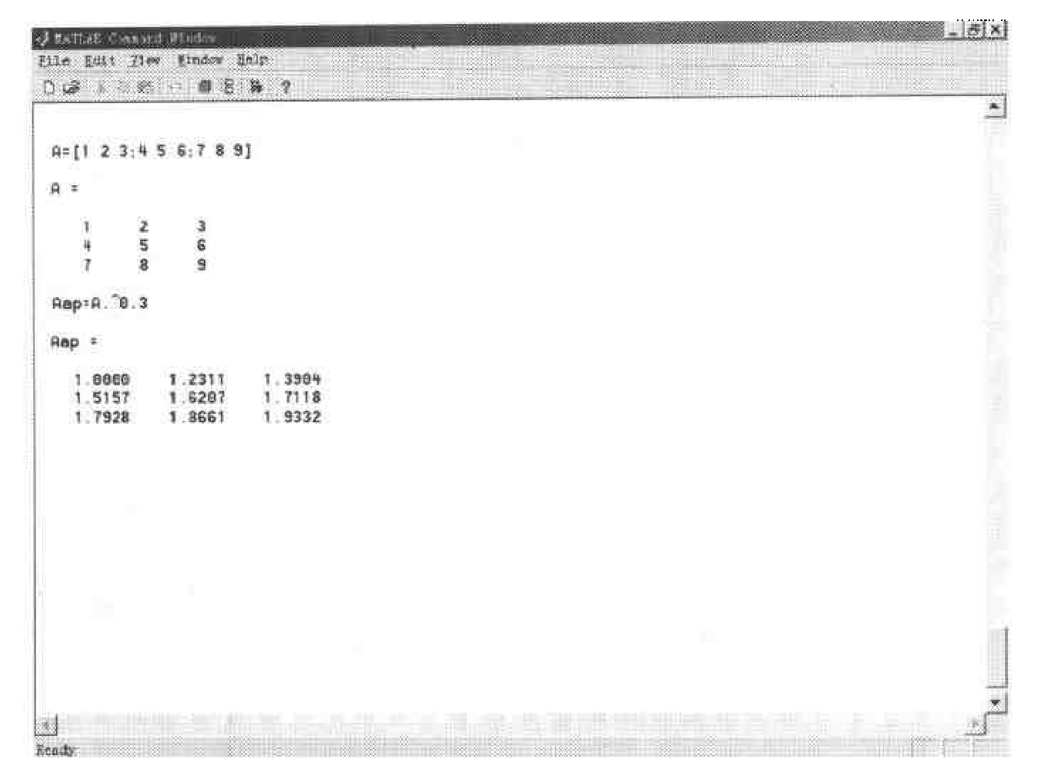


图 2.7 范例 2.7

2.2 矩阵与数组函数

MATLAB 所提供的函数有两类：一类是按数组运算法则设计的，称为数组函数，表示为 f(·)；另一类是按矩阵运算法则设计的，称为矩阵函数，表示为 funm(·).

2.2.1 基本数组函数

数组函数是按以下规则设计的：

对于 $A=\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$，定义 $f(A)=\begin{bmatrix} f(a_{11}) & \cdots & f(a_{1n}) \\ \vdots & \ddots & \vdots \\ f(a_{m1}) & \cdots & f(a_{mn}) \end{bmatrix}$

满足这个定义的基本函数指令被列于表 2.7 中.

表 2.7 基本数组函数表

函数名称	功能	函数名称	功能
sin	正弦	acosh	反双曲余弦
cos	余弦	atanh	反双曲正切
tan	正切	acoth	反双曲余切
cot	余切	asech	反双曲正割
sec	正割	acsch	反双曲余割
csc	余割	fix	朝零方向取整
asin	反正弦	ceil	朝正无穷大方向取整
acos	反余弦	floor	朝负无穷大方向取整
atan	反正切	round	四舍五入到整数

续表

函数名称	功　能	函数名称	功　能
atan2	四象限反正切	rem	除后取余数
acot	反余切	sign	符号函数
asec	反正割	abs	绝对值
acsc	反余割	angle	复数相角
sinh	双曲正弦	imag	复数虚部
cosh	双曲余弦	real	复数实部
tanh	双曲正切	conj	复数共轭
coth	双曲余切	log10	常用对数
sech	双曲正割	log	自然对数
csch	双曲余割	exp	指数
asinh	反双曲正弦	sqrt	平方根
bessel	第一、二类 Bessel 函数	erf	误差函数
beta	Beta 函数	erfinv	逆误差函数
gamma	Gamma 函数	ellipk	第一二类全椭圆积分
rat	有理近似	ellipj	Jacobi 椭圆函数

2.2.2 基本矩阵函数

MATLAB 所提供的基本矩阵函数指令被列于表 2.8 中.

表 2.8　基本矩阵函数表

函数指令	指　令　含　义	函数指令	指　令　含　义
cond(A)	A 阵的条件数(A 的最大奇异值除以最小奇异值)	svd(A)	A 阵的奇异值分解
det(A)	方阵 A 的行列式值	trace(A)	矩阵 A 的迹
dot(A)	矩阵的点积	expm(A)	矩阵指数 e^A
eig(A)	矩阵特征值	expm1(A)	用 Pade 近似求 e^A
norm(A,1)	A 阵的 1-范数	expm2(A)	用 Taylor 级数近似求 e^A,精度差
norm(A)	A 阵的 2-范数	expm3(A)	用特征值分解求 e^A,仅当独立特征向量数等于秩时适用
norm(A,inf)	A 阵的无穷范数	logm(A)	矩阵对数 ln(A)
norm(A,'fro')	A 阵的 F-范数	sqrtm(A)	矩阵 A 的平方根
rank(A)	A 阵的秩	kron(A,B)	Kronecker Tensor 乘积
rcond(A)	矩阵的倒条件数	funm(A,'FN')	A 阵的一般矩阵函数

例 2.8　求矩阵的行列式值、迹、秩和条件数, 如图 2.8 所示.

例 2.9　Kronecker Tensor 乘积是两个矩阵 ($A_{m\times n}$, $B_{p\times q}$) 的所有元素完全互乘得到的矩阵:

$$C=\begin{bmatrix} A_{1\times 1}*B & A_{1\times 2}*B & A_{1\times 3}*B & \cdots & A_{1\times n}*B \\ A_{2\times 1}*B & A_{2\times 2}*B & A_{2\times 3}*B & \cdots & A_{2\times n}*B \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ A_{m\times 1}*B & A_{m\times 2}*B & A_{m\times 3}*B & \cdots & A_{m\times n}*B \end{bmatrix}$$

```
A=magic(5)

A =

    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9

d=det(A),t=trace(A),rk=rank(A),c=cond(A)

d =

    5070000


t =

    65


rk =

     5


c =

    5.4618
```

图 2.8 范例 2.8

上述过程需调用 MATLAB 的 kron(A,B) 函数来实现，如图 2.9 所示.

```
A=fix(15*rand(3,3))

A =

     3     9    14
    10    11    13
    13    10     3

B=eye(3)

B =

     1     0     0
     0     1     0
     0     0     1

C=kron(A,B)

C =

     3     0     0     9     0     0    14     0     0
     0     3     0     0     9     0     0    14     0
     0     0     3     0     0     9     0     0    14
    10     0     0    11     0     0    13     0     0
     0    10     0     0    11     0     0    13     0
     0     0    10     0     0    11     0     0    13
    13     0     0    10     0     0     3     0     0
     0    13     0     0    10     0     0     3     0
     0     0    13     0     0    10     0     0     3
```

图 2.9 范例 2.9

例 2.10 向量 X 的 p 阶范数定义如下：

$$\| X \|_p = (\sum x_i^p)^{1/p}$$

其中 $p=1, 2\cdots, \infty$.

矩阵 A 的 p 阶范数定义如下：

$$\| A \|_p = \max_x \| Ax \|_p / \| x \|_p$$

其中 $p=1, 2\cdots, \infty$.

应用 MATLAB 的 norm (·) 函数做范数计算，如图 2.10 所示.

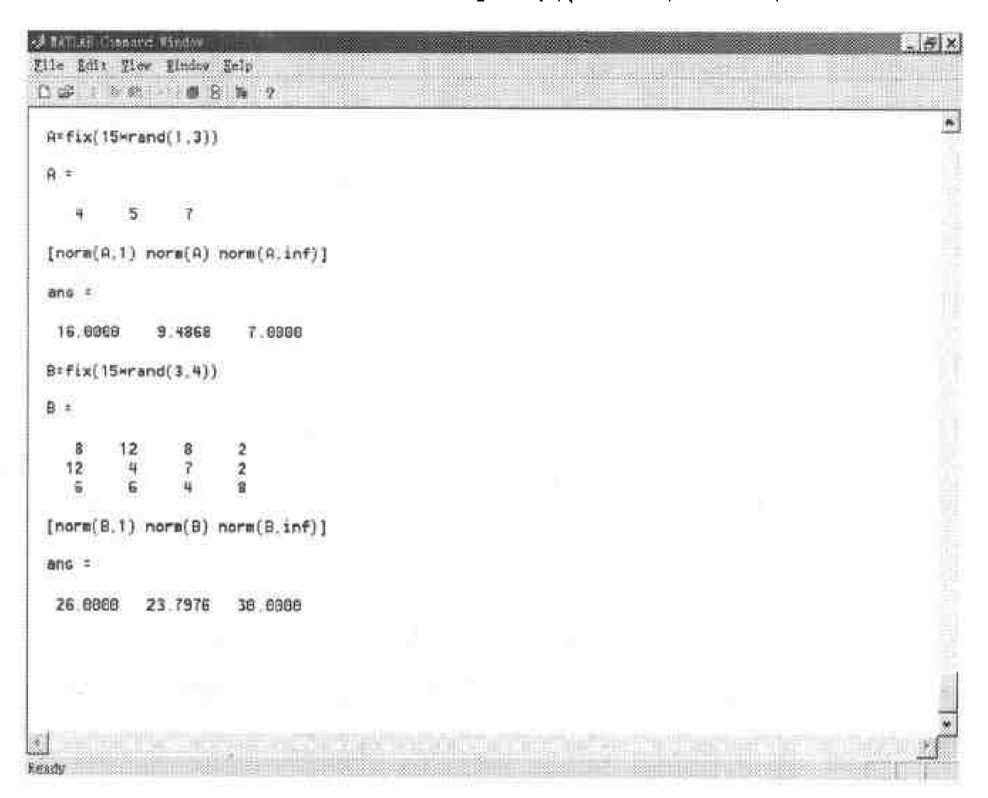


图 2.10　范例 2.10

2.2.3　几个易混淆的两种函数运算

在前面已经介绍了矩阵运算和数组运算之间的区别，本节进一步特别强调几种易引起混淆的矩阵和数组函数（表 2.9），并可从下面例子中体会.

表 2.9　几个易混淆的两种函数

数组函数	矩阵函数
exp (x)	expm (A)
log (x)	logm (A)
sqrt (x)	sqrtm (A)
	funm (A,'FN')

例 2.11　应用 sqrt 与 sqrtm 进行比较计算，如图 2.11 所示.

```
Bs =

   2.6458    3.1623
   3.8730    4.6904


BS =

   1.5867    1.1468
   2.6112    4.1779


BsBs =

  19.2474   23.1998
  28.4125   34.2474


BSBS =

   7.0000   10.0000
  15.0000   22.0000


BBS =

   7.0000   10.0000
  15.0000   22.0000
```

图 2.11 范例 2.11

例 2.12 计算 sin(B)和 funm(B,'sin')，如图 2.12 所示．

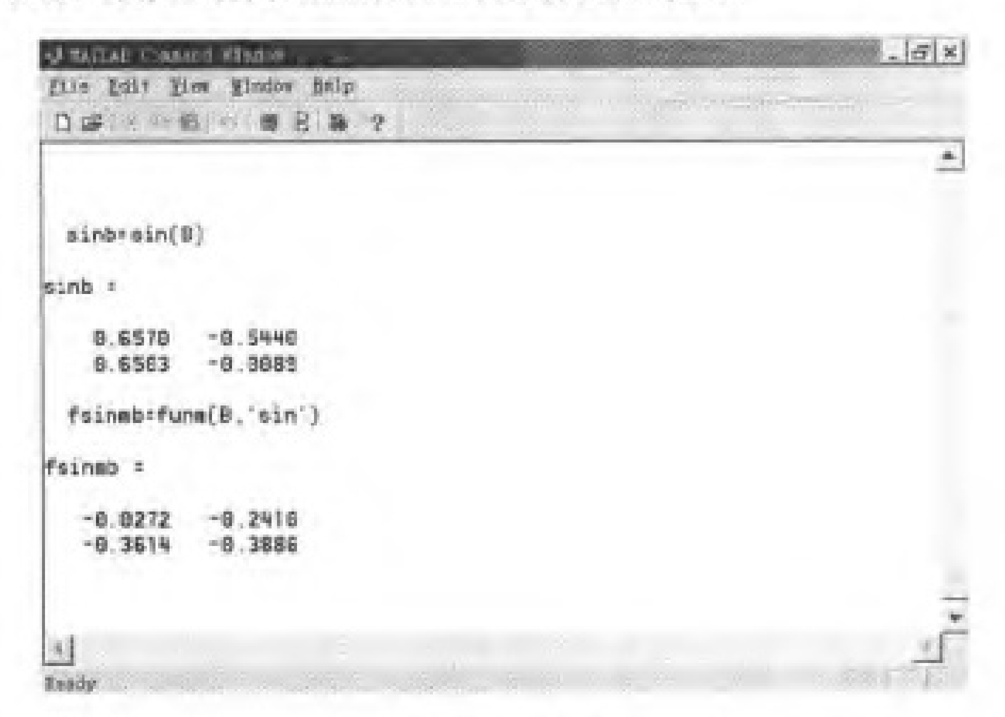


图 2.12 范例 2.12

2.3 关系运算和逻辑运算

MATLAB 所有的关系运算和逻辑运算均按数组运算规则定义．

2.3.1 关系运算

实现两个量之间比较的关系运算符有：

<（小于）、<=（小于等于）、>（大于）、>=（大于等于）、==等于）、~=不等于）.

关系运算法则说明：

1）当两个标量 a，b 比较时，则

若 a，b 间关系成立，那么关系运算结果为“1”；

若 a，b 间关系不成立，那么关系运算结果为“0”.

2）当对两个维数相同的数组 A 和 B 进行比较时，其比较是对 A 和 B 数组相同位置的元素按标量关系运算规则逐个进行，并给出元素比较结果，最终的关系运算的结果是一个维数与 A 或 B 相同，且其元素是由“0”或“1”组成的数组.

3）当标量 a 与数组 B 进行比较时，则是把标量 a 与 B 数组的每一个元素按标量关系运算规则逐个比较，并给出元素比较结果，最终的关系运算的结果是一个维数与 B 相同，且其元素是由“0”或“1”组成的数组.

4）在算术、关系和逻辑运算中，关系运算“优先权”居中.

例 2.13　为了简单说明关系运算符，给出图 2.13 所示的关系运算结果.

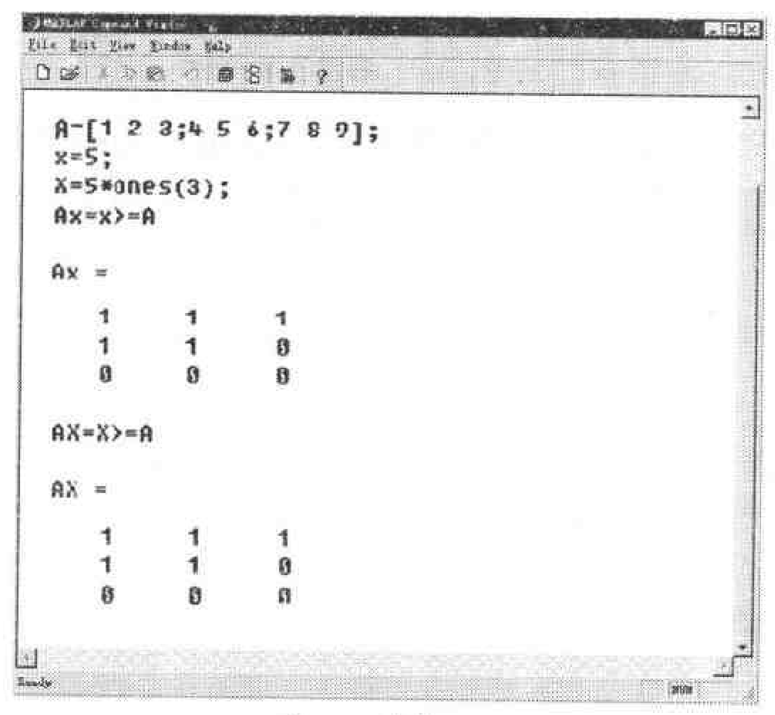


图 2.13　范例 2.13

2.3.2 逻辑运算

MATLAB 的逻辑运算有三种，即 &（或）、|（与）、~（非）. 与关系运算符类似，逻辑运算符是用来处理两个运算元，这两个运算元必须是相同大小的数组，其中一个也可以是常数.

逻辑运算法则说明：

1）在逻辑运算中，规定：

非零元素的逻辑量为“真”，用代码“0”表示；

零元素的逻辑量为“假”，用代码“1”表示．

2）若参与逻辑运算的是两个标量a和b，那么

a&b　　a，b全为非零时，运算结果为“1”；否则为“0”；

a|b　　a，b只要一个非零，运算结果为“1”；

~a　　当a是零时，运算结果为“1”，当a为非零时，运算结果为“0”．

3）若参与逻辑运算的是两个同维数组A，B．那么运算将对A，B相同位置上的元素按标量规则逐个进行．最终运算结果是一个与A（或B）同维的数组，其元素由“1”或“0”组成．

4）若参与逻辑运算的一个是标量a，一个是数组B，那么运算将在a与B中的每个元素之间按标量规则逐个进行．最终运算结果是一个与B同维的数组，其元素由“1”或“0”组成．

5）逻辑“非”是一元运算符，也服从数组运算规则．

6）在算术、关系、逻辑运算中，逻辑运算的运算“优先权”最低．

例 2.14　为了更好地理解逻辑运算符，给出图2.14所示的逻辑运算结果．

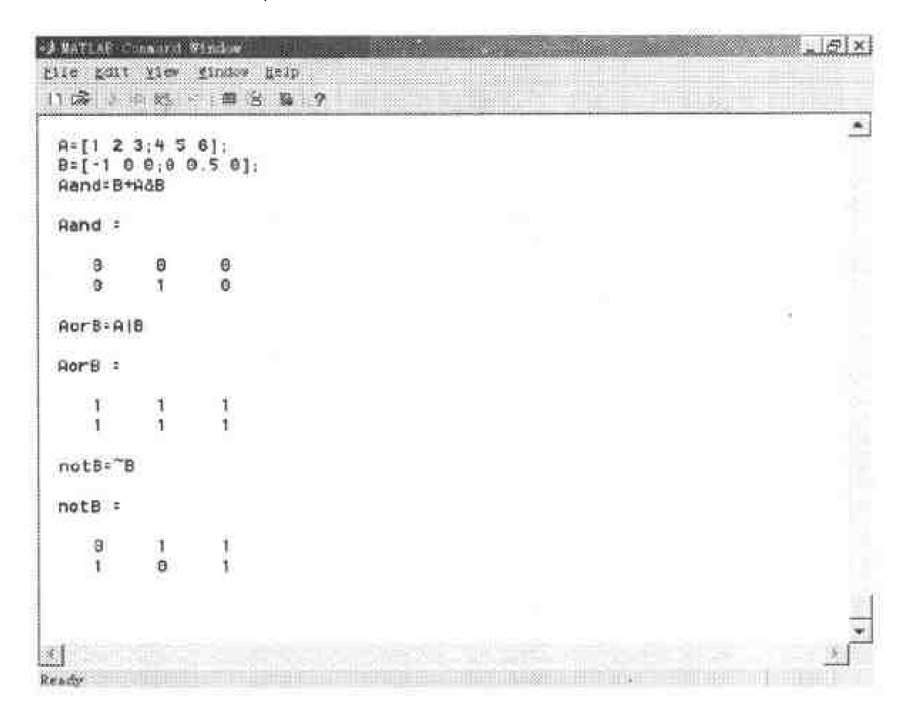


图2.14　范例2.14

除了逻辑运算符之外，MATLAB还提供了更为方便的逻辑函数．具体函数如下：

all（X）　　检查X是否全为TRUE；

isfinite（X）　　检查X是否全为有限值；

isnan（X）　　检查X是否为NaN；

isinf（X）　　检查X是否为无限大值；

xor（X，Y）　　执行X⊕Y运算．

2.4　矩阵的分解

MATLAB 矩阵的分解形式主要有：三角分解、正交化、奇异值分解和特征值分解．下面分别给予介绍．

2.4.1　三角分解

矩阵的三角分解是一个方阵分解为两个基本三角阵的乘积，其中一个三角阵为上三角阵，另一个为下三角阵．这种分解通常被称为“LU”分解，这当中使用的算法是高斯消元法．这种分解主要用于简化大矩阵行列式值的计算过程、求逆矩阵和求解联立方程组．不过应注意，这种分解法所得到的上下三角形矩阵并不是惟一的，还可以找到多个不同的上下三角形矩阵对，每对三角形矩阵相乘都会得到原矩阵．

例 2.15　对矩阵

$$A=\begin{bmatrix}1 & 2 & 3 & 4\\ 5 & 6 & 7 & 8\\ 9 & 10 & 11 & 12\\ 13 & 14 & 15 & 16\end{bmatrix}$$

进行LU 分解，然后再用逆运算证明 LU 分解的正确性，即由 L * U 生成A．其中 L 代表下三角形矩阵[1]，U 代表上三角形矩阵，调用“LU”函数的语法为[L,U]=lu(A)，如图 2.15 所示．

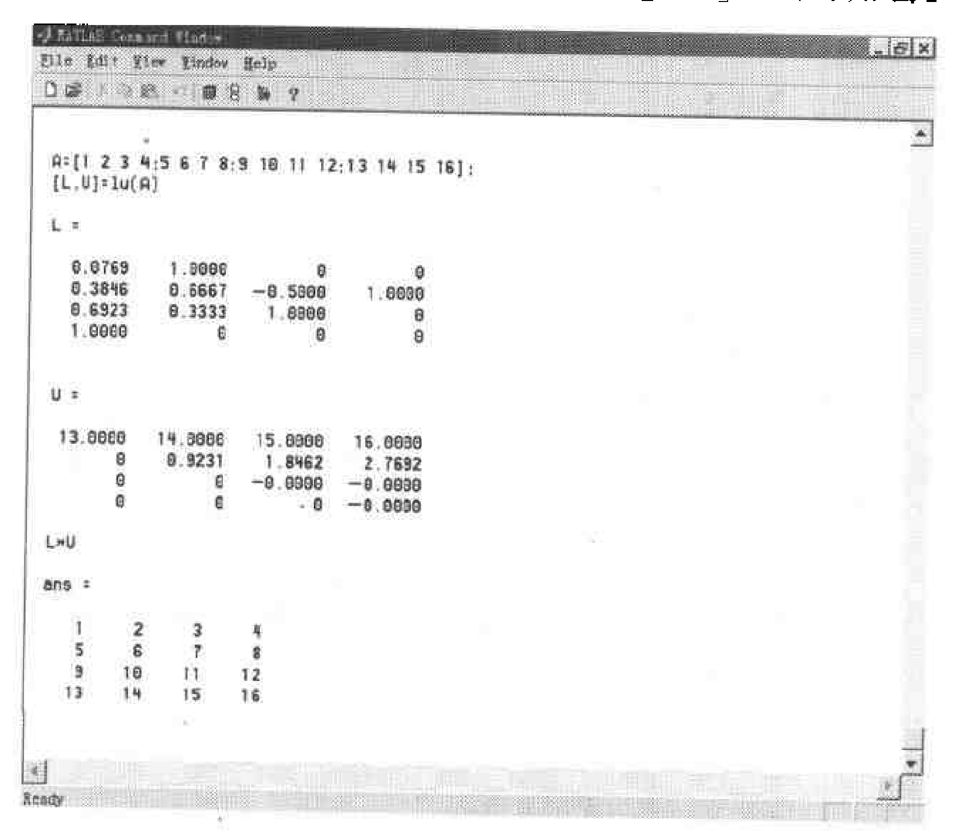


图 2.15　范例 2.15

1) L 是转换了的对角线为 1 的下三角矩阵．

2.4.2 正交分解

正交分解亦称QR分解，QR分解是将矩阵分解成一个单位正交矩阵与上三角形矩阵，对方阵和长方矩阵都很有用．MATLAB以qr函数来执行QR分解法，其语法为[Q，R]=qr（A），其中Q代表正规正交矩阵，而R代表上三角形矩阵．此外，原矩阵A不必一定是方阵，如果矩阵A为$m\times n$型，矩阵Q为$m\times m$型，则矩阵R为$m\times n$型．

例2.16 对矩阵

$$A=\begin{bmatrix}1 & 2 & 3 & 4\\ 5 & 6 & 7 & 8\\ 9 & 10 & 11 & 12\\ 13 & 14 & 15 & 16\end{bmatrix}$$

进行QR分解，然后再用逆运算证明QR分解的正确性，即由Q＊R生成A．如图2.16所示．

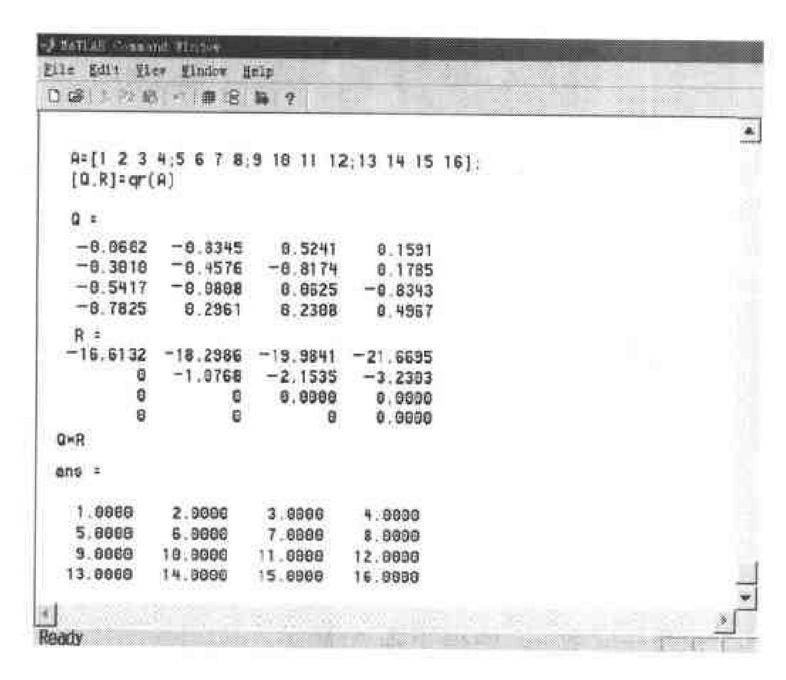


图2.16 范例2.16

2.4.3 特征值分解

如果A是$n\times n$矩阵，n个λ值满足式$Ax=\lambda x$，则λ为A的特征值，x为A的特征向量．

对于特征值问题，在MATLAB中可以用以下两种调用格式求A阵的特征值和特征向量：

d=eig(A)

[V,D]=eig(A)

其中，d 是 A 阵的特征值排成的列向量；D 是 A 阵的特征值的对角阵，V 是 A 阵的全部右特征向量构成的，并且 $AV=VD$.

例 2.17 演示简单实阵的特征值计算，如图 2.17 所示．

```
MATLAB Command Window
File  Edit  View  Window  Help

A=[0 1;-1 0]

A =

    0     1
   -1     0

eig(A)

ans =

        0 + 1.0000i
        0 - 1.0000i

[X,D]=eig(A)

X =

   0.7071              0.7071
        0 + 0.7071i         0 - 0.7071i


D =

        0 + 1.0000i         0
        0                   0 - 1.0000i

Ready
```

图 2.17　范例 2.17

2.4.4 奇异值分解

如果存在两个向量 u，v 及一常数 σ，使得矩阵 A 满足

$$AV=US$$

$$A'U=VS$$

由于 U，V 正交，所以可得奇异值表达式：

$$A=USV'$$

其中 S 是对角矩阵，也恰好是 A 阵的奇异值．

在 MATLAB 中，奇异值分解函数 svd()的调用格式为：[U,S,V]=svd(A)．

例 2.18 演示奇异值分解函数 svd()，如图 2.18 所示．

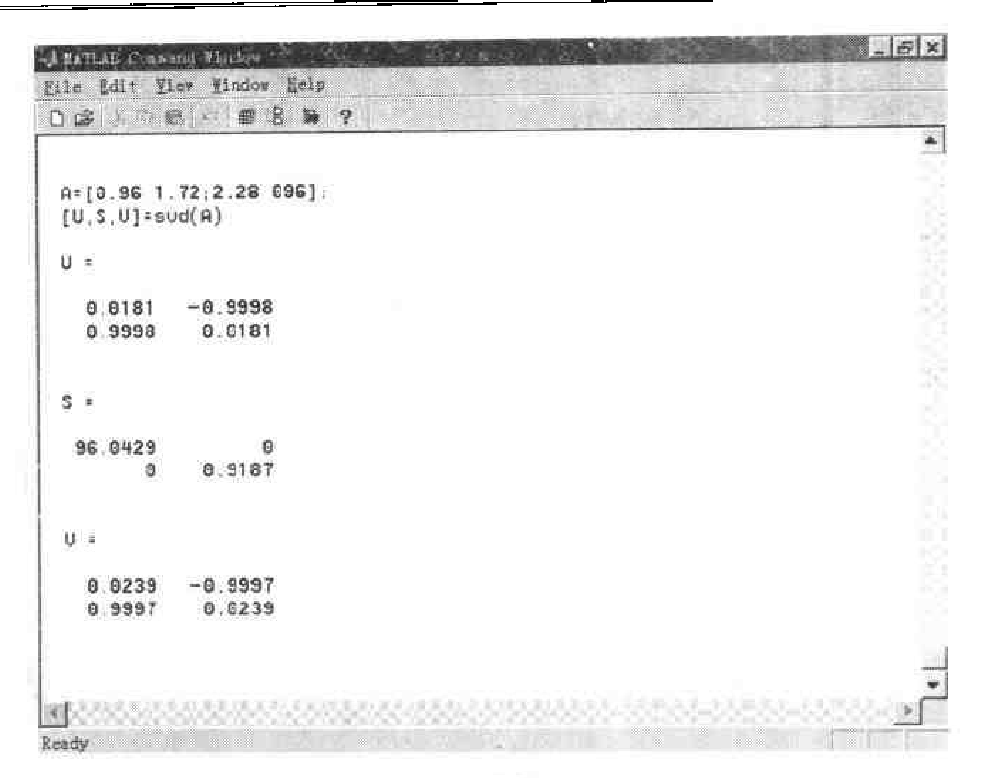


图 2.18　范例 2.18

2.5　多　项　式

2.5.1　多项式的表达和创建

在 MATLAB 中，对多项式表达方式约定为

$P(x)=a_0x^n+a_1x^{n-1}+\cdots+a_{n-1}x+a_n$ 用以下系数行向量表示：

$$P=[a_0\quad a_1\quad \cdots\quad a_{n-1}\quad a_n]$$

并且可利用指令：P=poly（AR），产生多项式系数向量．若 AR 是方阵，则多项式为特征多项式；若 AR 是向量，即 $AR=[ar_1\quad ar_2\cdots ar_n]$，则所得的多项式满足关系式：

$$(x-ar_1)(x-ar_2)\cdots(x-ar_n)=a_0x^n+a_1x^{n-1}+\cdots+a_{n-1}x+a_n$$

例 2.19　已知 A 阵，运用 poly 指令求它的特征多项式，具体程序和结果如图 2.19 所示．

注意：1）n 阶方阵的特征多项式系数向量一定是（$n+1$）维的．

2）特征多项式系数向量的第一个元素必是 1.

2.5.2　多项式的运算

多项式的乘、除运算分别与卷积、解卷机理完全相同．数学上卷积和解卷可分别用下述公式描述：

$$c(k)=\sum_{j=1}^{k}a(j)b(k+1-j)$$

$$c(k)-r(k)=\sum_{j=1}^{k}a(j)q(k+1-j)$$

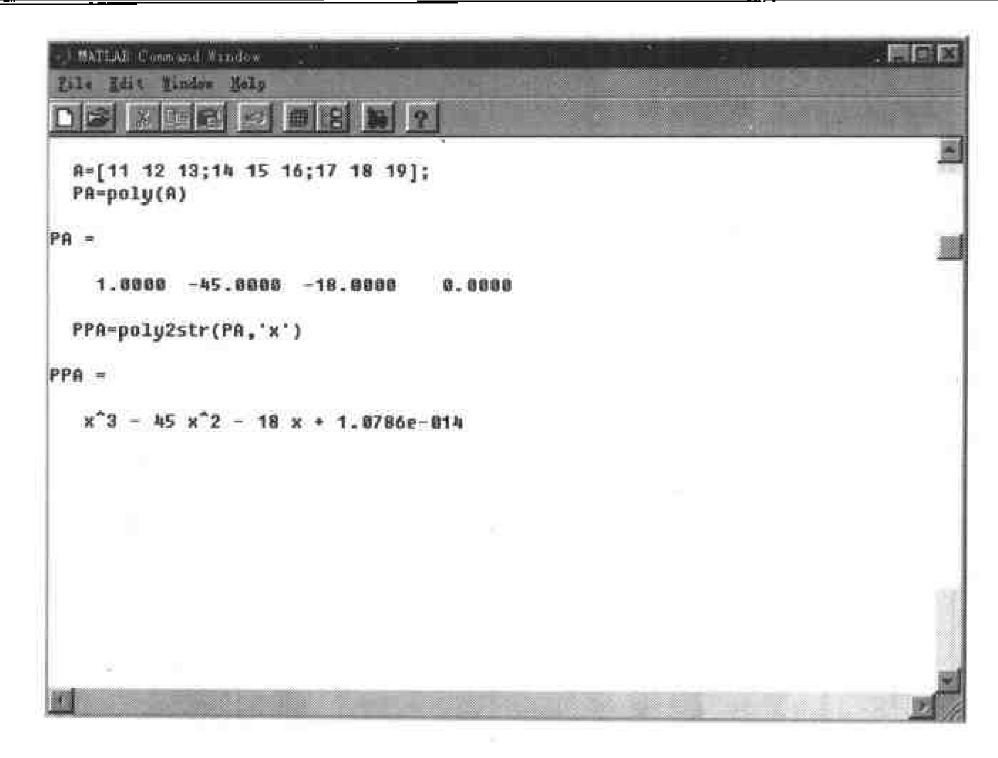


图 2.19 范例 2.19

在 MATLAB 中实现上述两种运算的指令分别是：c＝conv (a，b)、deconv (c，a)，它们的应用可从下面例子中进一步体会.

例 2.20 展开$(x^2+2x+2)(x+4)(x+1)$，然后再用$(x+4)$、$(x+1)$除验证其结果，如图 2.20 和图 2.21 所示.

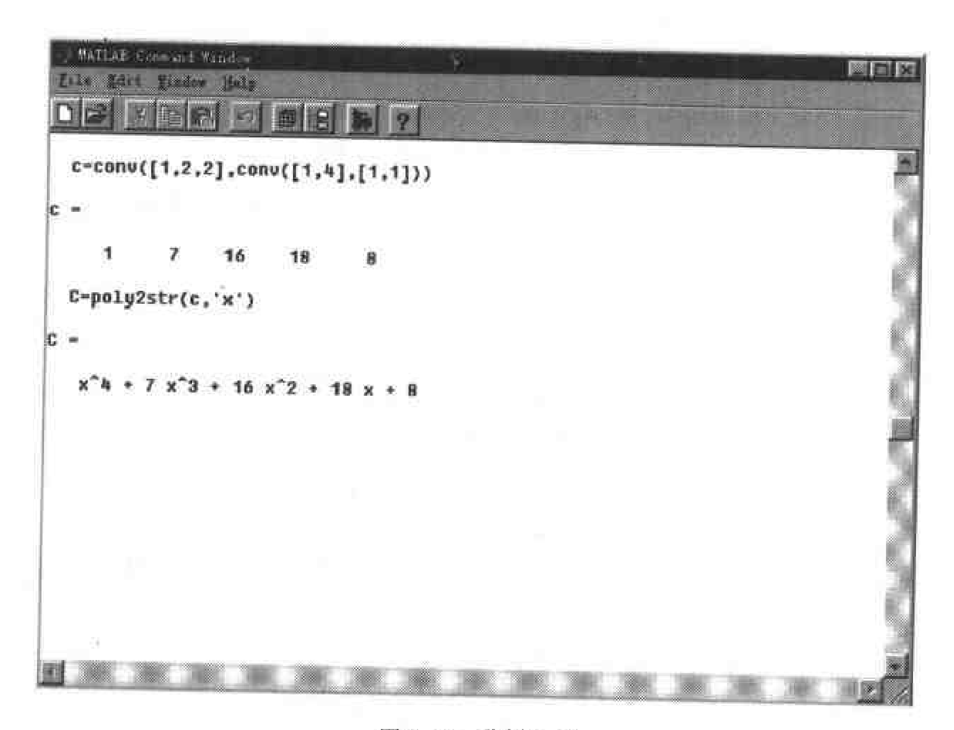


图 2.20 范例 2.20

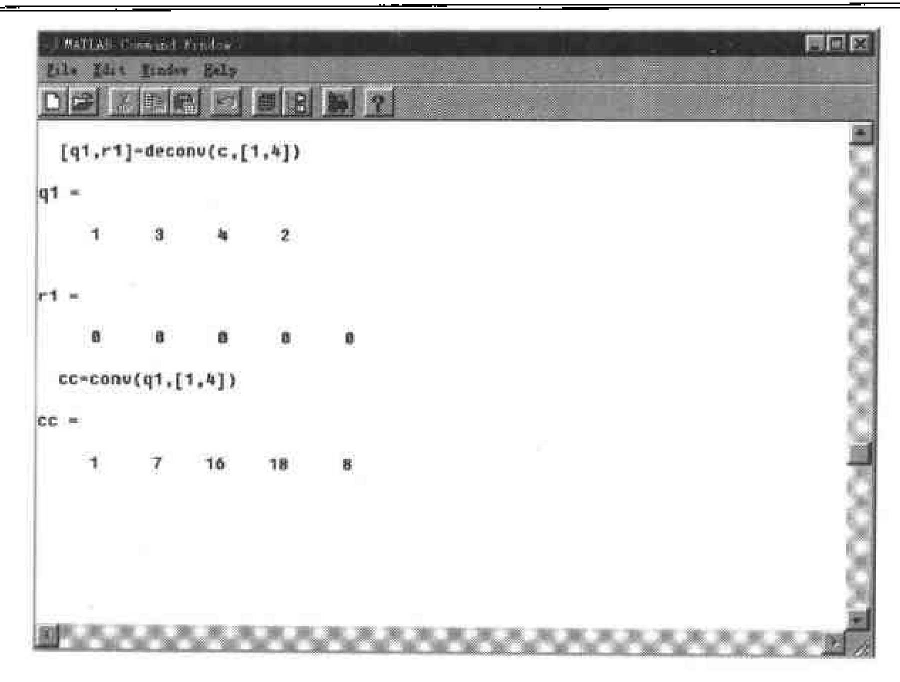


图2.21 范例2.20

2.6 数据分析

MATLAB 对数据分析指令的两条约定是：

1）若输入 X 是向量，那么不管是行向量还是列向量，运算是对整个向量进行的．

2）若输入 X 是数组（或称矩阵），那么指令运算是按列进行的．即默认每个列是由一个变量的不同“观察”所得的数据组成．

这两条约定不仅应用于本节所提到的指令，而且反映在 MATLAB 各工具包的软件中，为此，用户在编制自已的软件时，请尽量遵循以上约定，以便更好地利用 MATLAB 现有的函数指令．

2.6.1 基本统计函数指令

max（X） 找 X 各列的最大元素．

mean（X） 求 X 各列的平均值．

median（X） 找 X 各列的中位元素．

min（X） 找 X 各列的最小元素．

std（X） 找 X 各列的标准差．

prod（X） 找 X 各列元素之积．

sum（X） 找 X 各列元素之和．

S=cumsum（X） 求 X 各列元素累计和（可用于积分计算）．S 与 X 同维，且 $S_{ij}=\sum_{k=1}^{i} x_{kj}$．

P=cumprod（X） 求 X 各列元素累计积．P 与 X 同维，且 $P_{ij}=\sum_{k=1}^{i} x_{kj}$．

sort (X)　　使 X 各列元素按递增排序.

例 2.21　做统计计算，进一步理解上述函数，如图 2.22 所示.

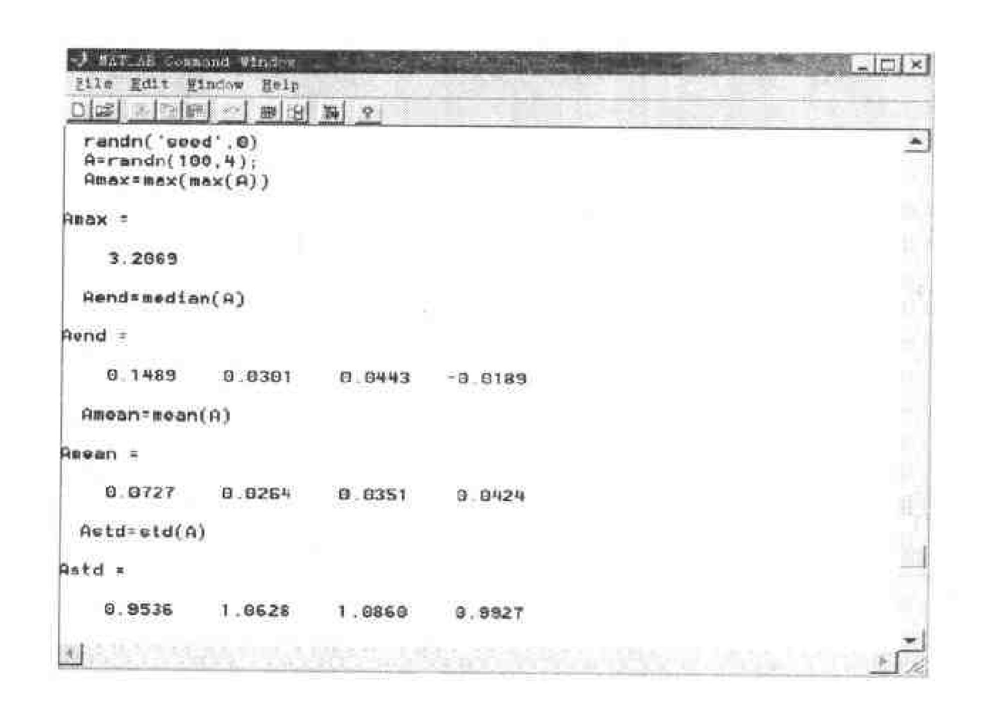


图 2.22　范例 2.21

2.6.2　协方差阵和相关阵

C=cov(X)　　求协方差阵 $\mathrm{cov}\{X\}=E\{(X-\mu_x)^T(X-\mu_x)\}$.

C=cov(x,y)　　求两随机变量的协方差$\begin{bmatrix}\sigma_x^2 & \mathrm{cov}(x,y)\\ \mathrm{cov}(y,x) & \sigma_y^2\end{bmatrix}$.

P=corrcoef(X)　　求相关阵，且 $p(i,j)=\dfrac{C(i,j)}{\sqrt{C(i,j)}\sqrt{C(i,j)}}$.

P=corrcoef(x,y)　　求两随机变量的相关系数(2×2).

例 2.22　运用上述函数对随机阵 X、Y 做统计计算，如图 2.23 所示.

例 2.23　X，Y 中的每个列都作为独立变量的记录看待时，求 X，Y 的互协方差阵，如图 2.24 所示.

2.6.3　有限差分和导数

DX=diff(X,k)　　按$(m\times n)$维 X 阵的列求$(m-k)\times n$ 维 k 阶差分矩阵.

V=del2(U)　　求$(m\times n)$维 X 阵的$(m\times n)$五点格式差分矩阵.

dyx=diff(Y)./diff(X)　　Y，X 是同维矩阵,差分计算按列进行.

dyx=gradient(Y,dx)　　Y 是向量,dx(缺省值为 1)是 X 的分度值.

[GX,GY]=gridient(Z,dx,dy)　　GX,GY 分别是二元函数的$\frac{\partial Z}{\partial x}$,$\frac{\partial Z}{\partial y}$.

Cyx=gridient(Z,dx,dy)　　复数阵 Cxy 的实、虚部分别是$\frac{\partial Z}{\partial x}$,$\frac{\partial Z}{\partial y}$.

```
MATLAB Command Window
File Edit Window Help
rand('seed',1)
X=rand(10,3);
Y=rand(10,3);
CX=cov(X)
CX =
   0.0588   -0.0159    0.0442
  -0.0159    0.0332    0.0006
   0.0442    0.0006    0.0882
CY=cov(Y)
CY =
   0.0870    0.0656   -0.0318
   0.0656    0.0874   -0.0576
  -0.0318   -0.0576    0.1013
Cxy=cov(X,Y)
Cxy =
   0.0577    0.0078
   0.0078    0.0877
Pxy=corrcoef(X,Y)
Pxy =
   1.0000    0.1104
   0.1104    1.0000
```

图 2.23　范例 2.22

```
MATLAB Command Window
File Edit Window Help

xbar=mean(X);[mx,nx]=size(X);
ybar=mean(Y);[my,ny]=size(Y);
CXY=(X-ones(mx,1)*xbar)'*(Y-ones(my,1)*ybar)/(mx-1)

CXY =

  -0.0120    0.0001   -0.0039
   0.0206    0.0139    0.0066
  -0.0068   -0.0252    0.0148

xu=sqrt(diag(CX));
yu=sqrt(diag(CY));
PXY=CXY./(xu*yu')

PXY =

  -0.1809    0.0016   -0.0537
   0.3828    0.2576    0.1140
  -0.0782   -0.2867    0.1567
```

图 2.24　范例 2.23

说明

1）五点格式差分算法源于拉普拉斯方程$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=0$和泊松方程$\Delta^2 u=\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}$的差分解法．

2）在求偏导数指令中，若输入量dx，dy取标量值．则它们表示求近似偏导数时自变量的分度值．若输入量dx，dy取向量值，则它们分别表示求近似偏导数的自变量绝对

坐标值．dx 的长度与 Z 的列维相同，dy 的长度与 Z 的行维相同．

3）在求偏导数指令中：GX 与 Z 同维，它给出在自变量绝对坐标值所决定网点上的 $\frac{\partial Z}{\partial x}$ 偏导数值；GY 也与 Z 同维，它给出在自变量绝对坐标值所决定网点上的 $\frac{\partial Z}{\partial y}$ 偏导数值．

例 2.24　按列求差分，如图 2.25 所示．

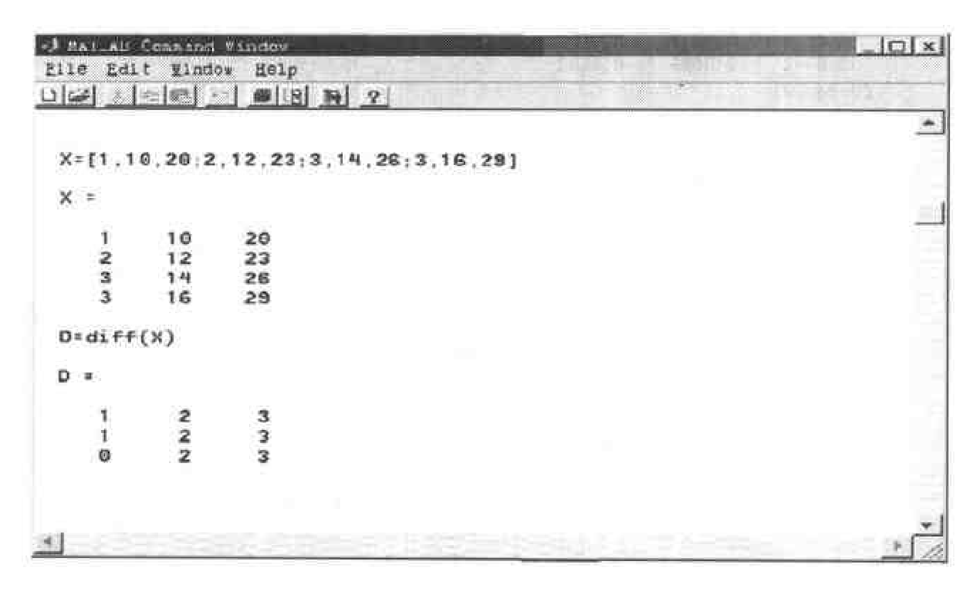


图 2.25　范例 2.24

例 2.25　五点格式差分（$u=x^2+y^2$，$\Delta^2 u=4$），如图 2.26 所示．

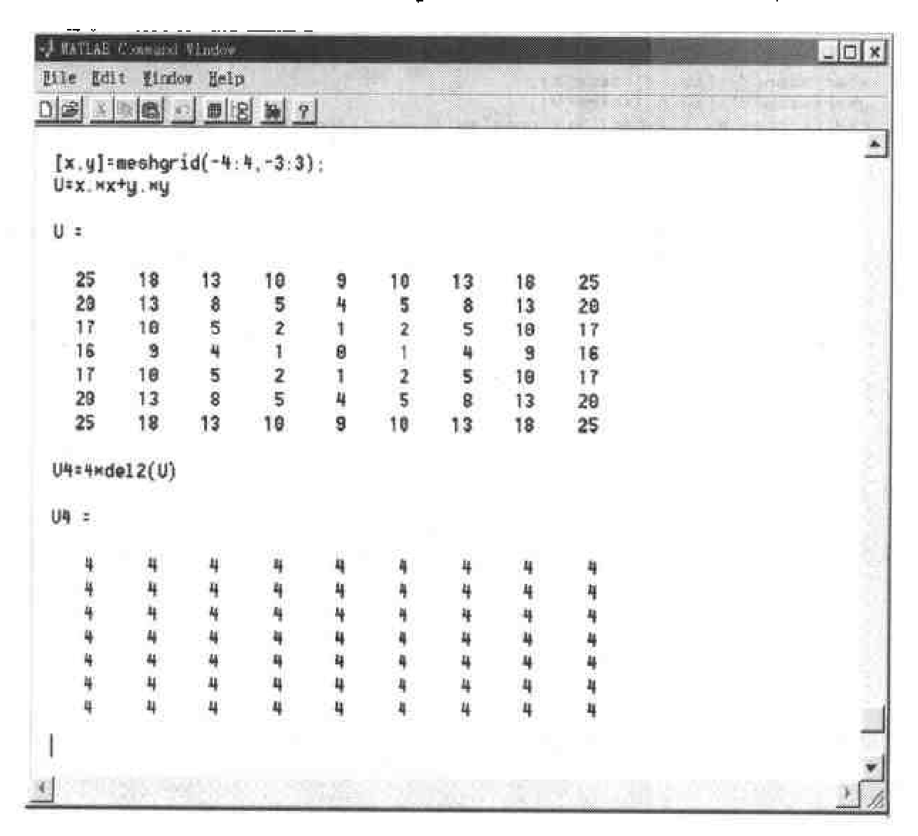


图 2.26　范例 2.25

2.6.4 数据滤波

Y=FILTER (B, A, X) 数字滤波

说明：

该滤波器的数学模型为

a(1) * y(n)=b(1) * x(n)+b(2) * x(n−1)+…+b(nb+1) * x(n−nb)−a(2) * y(n−1)−…−a(na+1) * y(n−na)

例 2.26 如图 2.27 所示数字滤波过程的指令示范.

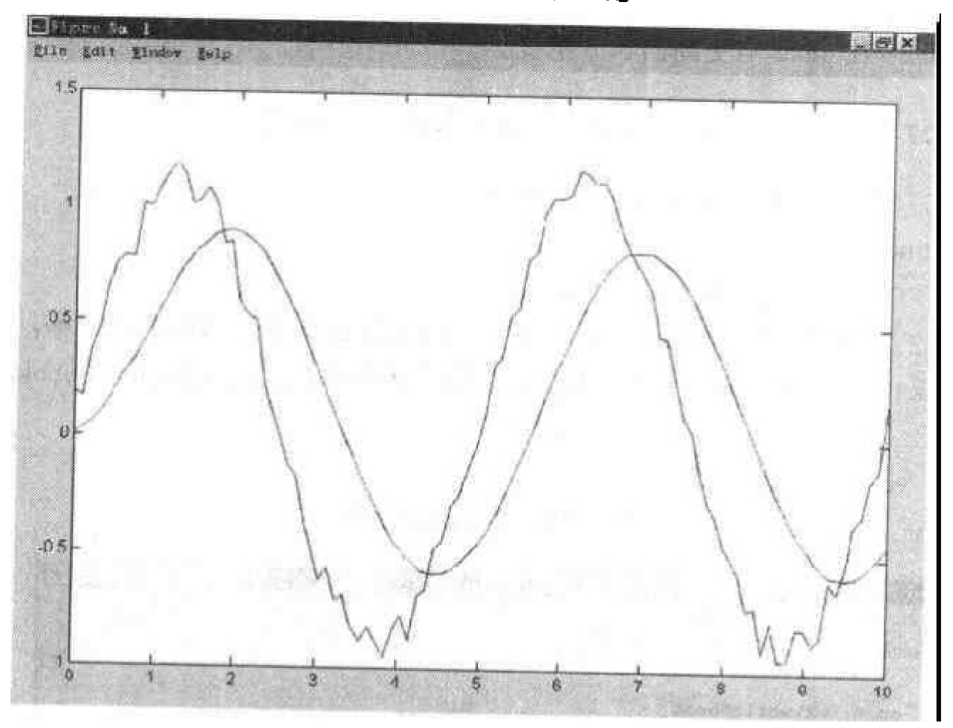


图 2.27 范例 2.26

```
t=linspace(0,10,100);
s=sin(2*pi/5*t);
noise=0.2*rand(size(t));
x=s+noise;
y=zeros(size(x));
A=[1 -0.9];
B=[0.05 0.06];
y=filter(B,A,x);
plot(t,x,'b',t,y,'r')
```

2.7 数值分析

2.7.1 数值积分

(1) 求函数的数值积分

Q=QUAD ('F', A, B, TOL, TRACE)　　自适应递推辛普生 (Simpson) 法求

数值积分.

Q=QUAD8（'F'，A，B，TOL，TRACE）　　自适应递推牛顿-柯西（Newton-Cotes）法求数值积分.

说明：

1）第一个输入参数'F'是被积函数表达式字符串或函数文件名.

2）第二三个输入参数 A、B 分别是积分的上、下限.

3）输入参数 TOL 用来控制积分精度. 缺省时，默认 tol=0.001.

4）输入参数 trace，若取 1 则用图形展现积分过程，取 0 无图形. 缺省时，不画图.

5）quad8 比 quad 有更高的积分精度.

例 2.27　已知 $y(t)=e^{-0.5t}\sin(t+\frac{\pi}{6})dt$，求 $S=\int_0^{3\pi} y(t)dt$.

步骤 1　用编辑器建立被积函数 esin.m.

```
function y=esin(t)
y=exp(-0.5*t)*sin(t+pi/6);
```

步骤 2　把写好的 esin.m 文件存放于用户自己的工作目录（如：d:\wx\matlabbook），然后在 MATLAB 环境下，利用下述命令使 d:\wx\matlabbook 成为当前工作目录.

```
cd d:\wx\matlabbook
```

步骤 3　调用积分指令，运行结果如图 2.28 所示.

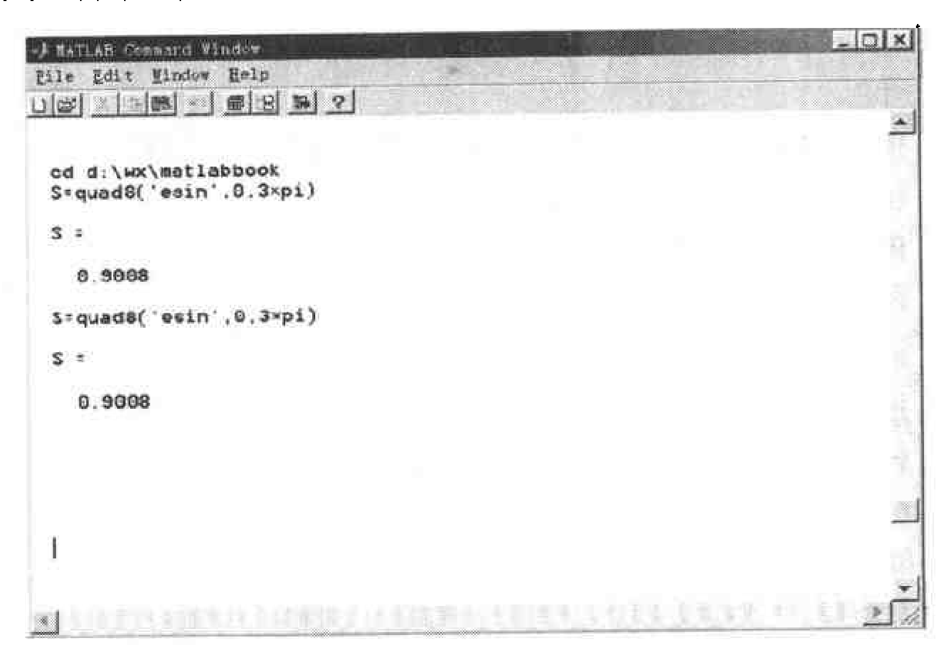


图 2.28　范例 2.27

（2）通过采样向量求数值积分

Z=TRAPZ（X，Y）　　用梯形法求数值积分.

Sc=CUMSUM（X）　　用欧拉法求积分函数值.

说明：

1）在 TRAPZ 中，当 X，Y 是同维的列（行）向量时，所得的 Z 将是标量，它给出

Y 相对 X 的积分值；当 Y 是（$m\times n$）维矩阵时，X 必须是（$m\times 1$）的列向量，积分对 Y 各列分别进行，计算所得（$1\times n$）维 Z 的元素是对应列函数对于 X 的积分．在该指令中，X 缺省时，认为 Y 为单位步长等距采样所得的函数阵．

2）CUMSUM 的运算结果 Sc 与 X 同维．Sc 的各列给出 X 相应列的积分函数值，且认为积分采用的是等矩单位步长．一般说来，该积分的精度较差．



例 2.28　用梯形法和欧拉法求例 2.27 中的数值积分，结果如图 2.29 所示．

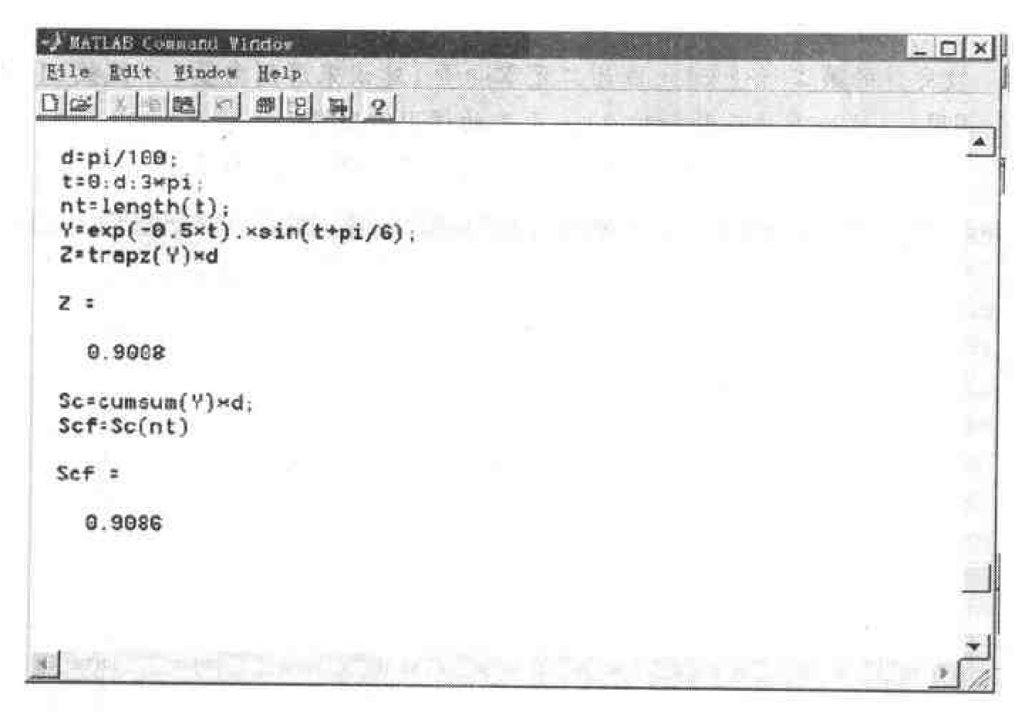


图 2.29　范例 2.28

2.7.2　微分方程的数值解

[t,x]=ODE23('F',TSPAN,x0)

[t,x]=ODE45('F',TSPAN,x0)

说明：

1）两个指令的调用格式完全相同，均采用 Runge-Kutta 法．

2）该指令是对一阶常微分方程组设计的．因此，假如待解的是高阶微分方程，那么它必须先被转化为形如 $\dot{x}=f(x,t)$ 的一阶微分方程组，即"状态方程"．

3）第一个输入参数'F'是定义 $f(t,x)$ 的函数文件名．该函数文件必须以 $\dot{x}$ 为输出，以 t，x 为输入．

4）TSPAN=［T0 TFINAL］，T0，TEINAL 分别是积分的起始值和终止值．

5）一般说来，ode45 比 ode23 的积分分段少，而运算速度更快．

例 2.29　求著名的 Van der Pol 方程 $\ddot{x}(x^2-1)\dot{x}+x=0$.

步骤 1　转化为状态方程．

令 $x_1=\dot{x}, x_2=x$ 那么可把 $\ddot{x}+(x^2-1)\dot{x}+x=0$ 写成状态方程形式：

$$\begin{cases}\dot{x}_1=(1-x_2^2)x_1-x_2\\ \dot{x}_2=x_1\end{cases}$$

步骤 2 建立函数文件 xprime.m.

```
function xdot=xprime(t,x)
xdot=zeros(2,1);
xdot(1)=(1-x(2)^2)*x(1)-x(2);
xdot(2)=x(1);
```

说明 a. 注意函数文件第一行：函数的输入参数一定是两个；次序一定是先“时间变量”，后“状态变量”.

b. 函数文件的第二行使xdot 成为二元零向量（建议采用列向量，以便被MATLAB其他指令调用），这行是为后两行对 xdot 元素的调用做准备.

步骤 3 解微分方程．其程序和程序的运行结果如图 2.30 所示．

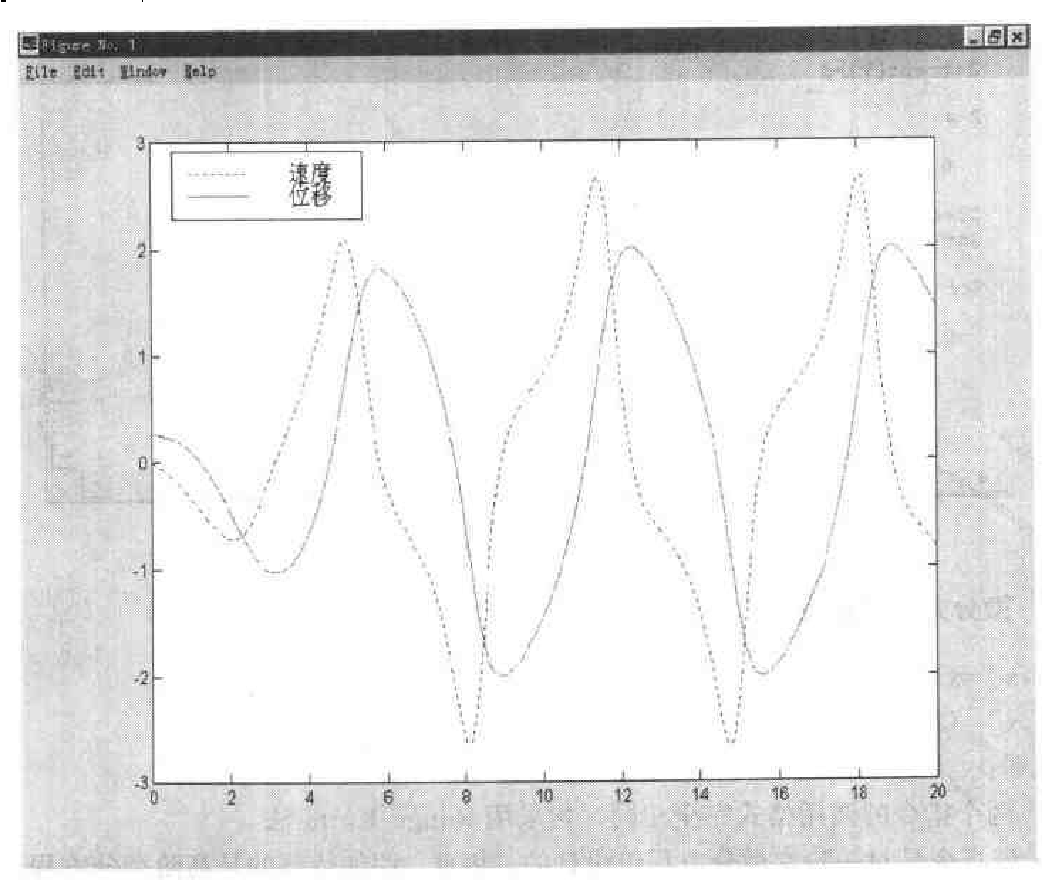


图 2.30 范例 2.29

```
TSPAN=[0,20];
x0=[0,0.25]';
[t,x]=ode23('xprime',TSPAN,x0);
plot(t,x(:,1),':b',t,x(:,2),'-r')
legend('速度','位移')
```

第三章　MATLAB 符号处理

除了数值计算以外，在各个领域中还经常遇到符号计算问题．所谓符号计算，就是指运算对象过程允许存在非数值的符号变量．符号计算有两个特点：第一是运算对象和过程允许存在非数值的符号变量；第二是可以获得任意精度的数值解．下面举一些简单的实例让读者体会一下符号计算的这两个特点．

例 3.1　求函数 $f(x)=\sin^2(x)$ 的微积分，如图 3.1 所示．

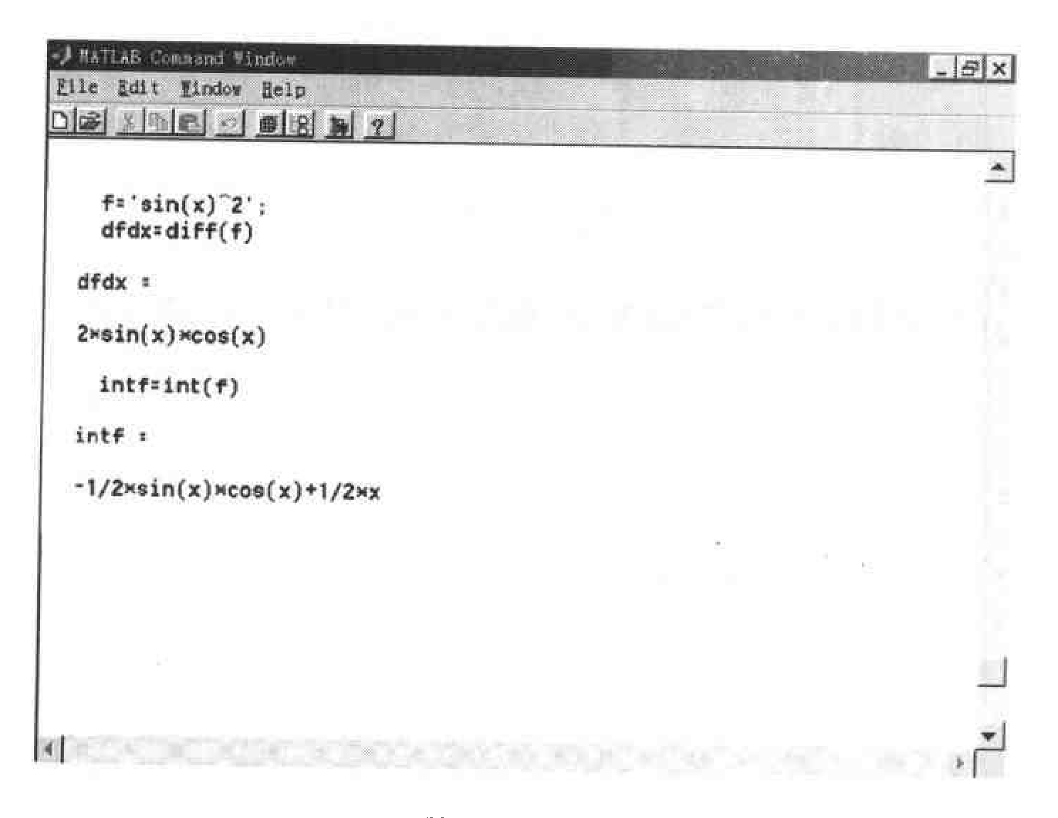


图 3.1　范例 3.1

例 3.2　求矩阵的行列式值、逆、特征值，如图 3.2 所示．

至于怎样以精确解获得任意精度解，怎样改变默认精度，怎样把任意精度符号解变成“真正”数值解，则需要运用以下指令实现．

digits (n)　　使该指令后的近似解的精度为 n 位有效数字．

subs (S, Dsym, Fsym)　　把 S 符号解中的自由参数 Fsym 用数字字符 Dsym 代替．

vpa (S, n)　　求 S 的 n 位有效数字近似解，n 缺省时，给出默认精度近似解．

numeric (S)　　把不含自由参数的 S 符号变量转化为数值变量．

例 3.3　求方程 $x^2-x-1=0$ 的精确解和任意精度，如图 3.3 所示．

```
MATLAB Command Window
File Edit Window Help

  A=sym('[a11,a12;a21,a22]');      %定义矩阵A
  inv(A)                           %求矩阵A的逆

ans =

[ -a22/(-a11*a22+a12*a21),  a12/(-a11*a22+a12*a21)]
[  a21/(-a11*a22+a12*a21), -a11/(-a11*a22+a12*a21)]

  determ(A)                        %求矩阵A的行列式

ans =

a11*a22-a12*a21

  eigensys(A)                      %求矩阵A的特征根

ans =

[ 1/2*a11+1/2*a22+1/2*(a11^2-2*a11*a22+a22^2+4*a12*a21)^(1/2)]
[ 1/2*a11+1/2*a22-1/2*(a11^2-2*a11*a22+a22^2+4*a12*a21)^(1/2)]
```

图 3.2　范例 3.2

```
MATLAB Command Window
File Edit Window Help

  R1=solve('x^2-x-1=0');
  RV=vpa(R1)

RV =

[  1.6180339887498948482045868343657]
[ -.6180339887498948482045868343656]

  RV4=vpa(R1,4)

RV4 =

[  1.618]
[ -.6180]

  RV20=vpa(R1,20)

RV20 =

[  1.6180339887498948482]
[ -.61803398874989484820]
```

图 3.3　范例 3.3

例 3.4　对于 $f(x)=\sin^2(a*x)$，求 $\frac{d}{dx}f(x)$ 各种表现形式的精确解，如图 3.4 所示．

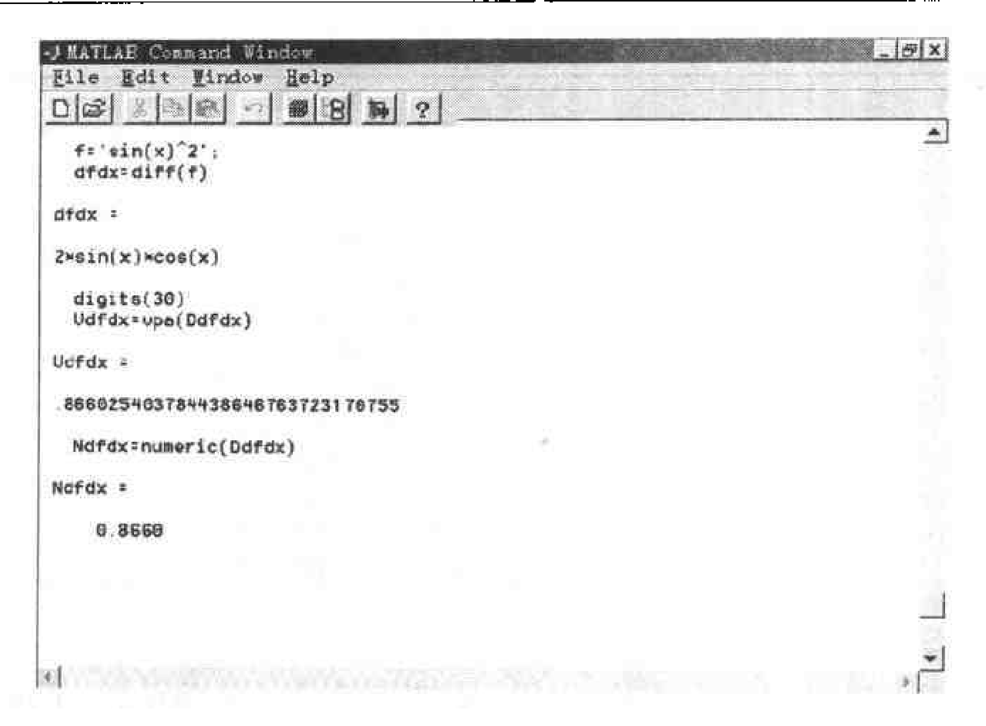


图 3.4 范例 3.4

3.1 字 符 串

本节内容包括如何通过建立字符数组或单位数组来表示字符串，并讨论字符串的查找、替换和如何做字符串与数值间的转换等.

字符串的常用函数及其功能如表 3.1 所示.

表 3.1 字符串的常用函数

函 数	功 能
char	创建字符数组（字符串）
double	将字符串转换为数值
cellstr	从字符数组中创建单元数组
blanks	创建空白字符串
deblank	删除字符串后面的空白
eval	执行字符串所代表的指令
ischar	判断是否为字符串数组
iscellstr	判断是否为字符单元数组
isletter	判断字符是否为字母
isspace	判断字符是否为空白字符
strcat	连接水平排列的字符串
strvcat	连接垂直排列的字符串
strcmp	字符串的比较
strncmp	字符串前 N 个字符的比较
findstr	查找字符串
striust	对齐字符串
strmatch	查找相同字符串
strrep	字符串的替换

续表

函　数	功　能
strtok	查找字符串的结束字符
upper	将字符串转换为大写
lower	将字符串转换为小写
num2str	将数值转换成字符串
int2str	将整数转换成字符串
mat2str	将数组转换成字符串
str2num	将字符串转换成数值
prinstf	将字符串写入特定格式
dec2hex	将十进制转换为十六进制
bin2dec	将二进制转换为十进制
dec2bin	将十进制转换为二进制
base2dec	将字符串按照一定的基数转换为十进制
dec2base	将十进制按照 ·定的基数转换为字符串

3.1.1 字符数组

在MATLAB中，字符串可以用字符数组来表示，而字符数组则与ASCII码相对应.一个字符是由两个字节构成的．可以举例说明，命名一个包含12个字符串为“name”：

```
name='Thomas R.Lee';
```

查看name的内容：

```
whos
  Name        Size        Bytes Class
  name        1×12        24 char array
  Grand total is 12 elements using 24 bytes
```

可以发现name这个字符串占用了24个字节．

用class及ischar这两个函数来判断“name”中保存的数据是否是字符：

```
class (name)
ans=
     char
ischar (name)
ans=
     1
```

3.1.2 字符的ASCII码转换

字符数组中的每个字符是由16位的ASCII码所组成，可以利用double这个函数来将字符串转换成它所代表的ASCII码；也可以利用char这个函数将ASCII码转换成原来的字符．

以上面所述的“name”为例：

```
name=double (name)
name=
        84  104  111  109  97  115  32  82  46  76  101  101
```

```
name=char (name)
name=
        Thomas R. Lee
```

3.1.3 创建二维的字符数组

在创建一个二维字符数组前，先要确定数组的每一行字符个数都必须相等．举例来说，下面的例子是一个正确的字符数组，数组中的每一行都有 12 个字符：

```
name= ['Thomas R. Lee';'Sr. Developer']
name=
        Thomas R. Lee
        Sr. Developer
```

如果要建立每行字符数不相等的数组时，可以利用输入空白字符将数组中字符数不足的行补成与其他行相等就可以了；另外也可以利用char 函数自动创建每行字符数都相等的数组．char 会自动按照数组中所有的字符串查找字符数最多的那一行，而后以加入空白字符的方式来使数组中的所有行字符数相等．

下面举例来说明：

```
name=char ('Thomas R. Lee','Senior. Developer')
name=
        Thomas R. Lee
        Senior. Developer
```

如果需要得到数组中字符串的实际内容时，可以利用 deblank 函数来删除字符串后的空白字符．如：

```
trimname=deblank (name (1,:))

trimname=
            Thomas R. Lee
size (trimname)
ans=
      1     12
```

3.1.4 字符串中的单元数组

通常利用单元数组来保存字符串的数据会比字符数组方便．这是因为使用单元数组可以免去建立一个字符数组所需要的“数组中每行的字符数都必须相等”的规定，因而使用起来更为便利．

在下列情况下，可以使用 MATLAB 中关于单元矩阵的函数来完成操作：

1）字符串中的字符数组与单元数组间的转换；

2）利用单元数组进行字符串的比较．

3.1.5 字符数组与单元数组间的转换

可以利用 cellstr 函数将字符数组转换为单元数组，例如：

```
data= ['Allison Jones';'Development';'Phoenix ']
data=
        Allison Jones
        Development
        Phoenix
```

在这个字符数组中，每一行的字符数都相等．我们还可以利用 cellstr 建立一个单元所组成的行，每个单元包含--个字符串，例如：

```
celldata=cellstr (data)
celldata=
            'Allison Jones'
            'Development'
            'Phoenix'
```

通过运行 length 函数，可以发现 cellstr 函数会删掉每行后面的空白字符，如：

```
length (celldata {2})
ans=
     11
length (celldata {3})
ans=
     7
```

可以利用 char 函数将 celldata 转换回原来的字符数组，如：

```
strings=char (celldata)
strings=
          Allison Jones
          Development
          Phoenix
```

3.1.6 字符串比较

对两个字符串或字符串中的部分字符串进行比较，一般有下列几种情况：

1) 比较两个字符串或两个字符串中的子字符串是否相等；
2) 比较两个字符串中的个别字符是否相等；
3) 先将字符串分为几个部分，判断每个部分是否为空白字符．

3.1.7 判断字符串是否相等

MATLAB 中有两个函数可以用来判断两个输入的字符串是否相等：

strcmp：比较两个输入字符串是否相等；

strncmp：比较两个输入字符串的前几个字符是否相等．

现在举例说明如下：

```
str1='hello';
str2='help';
c=strcmp (str1, str2)
c=
    0
```

str1 与 str2 这两个字符的内容很明显不一样，如用 strcmp 函数来比较这两个字符串，结果会返回一个 0 值，表示字符串的内容不同．如果用 strncmp 来比较这两个字符串的前三个字符，结果字符都相同，所以会返回 1，例如：

```
c=strncmp (str1, str2, 3)
c=
    1
```

3.1.8　通过字符的运算来比较字符

只要字符数组拥有相同的维数，就可以利用 MATLAB 中的运算法则对字符数组进行比较，例如：

```
A='fate';
B='cake';
A==B
ans=
     0    1    0    1
```

上面是用"=="来比较两个字符中的字符是否相等．A 与 B 这两个字符串中第 2 个及第 4 个字符相同，所以返回值为 1，如果字符不相同，则返回 0．类似地还可以用其他的运算符号（如<，<=，>，>=，! =等）做字符数组运算，这些运算符号会根据字符所对应的 ASCII 码比较字符之间的关系．

3.1.9　字符串中字符的分类

在 MATLAB 中可以用两个函数来做字符串中字符的分类：

1）使用 isletter 来判定字符串中字符的分类；

2）使用 isspace 来判定字符串中的字符是否是空白字符．

举例来说，先创建一个字符串：

```
mystring='Room410';
```

然后使用 isletter 来判定字符串中的字符是否为字母：

```
A=is  letter (mystring)
A=
   1    1    1    1    0    0    0
```

这个字符串中的前 4 个字符是字母组成的，所以会返回 1，而后 3 个字符不是字母，所以会返回 0.

3.1.10 查找与替换

MATLAB 提供了几个函数，可以对字符串中的字符做查找和替换操作．举例来说，先创建一个字符串：

```
mystring='Sample 1, 12/22/99';
```

可以利用 strrep 函数来做字符串中的字符的查找及替换操作，例如将“12/22”替换为“12/20”，例如：

```
newstring=strrep (mystring,'22','20')
newstring=
            Sample 1, 12/20/99
```

findstr 函数会根据所给的字符串中的字符来做字符串的查找，当查找成功后会返回第一个相同字符的位置，例如：

```
position=findstr ('amp', mystring)
position=
          2
```

要查找的字符串是“amp”，与'Sample 1，12/22/99'中的第二个字符开始连续相同，所以会返回开始相同字符的位置值 2.

可以使用 strtok 函数将一个字符串中某个特定的字符前面的字符分离出来，其中特定字符省略时的默认值是空白字符，例如：

创建一个字符串：

```
string='good morning! bye-bye! ';
```

应用 strtok 函数，体会运行后结果：

```
strtok (string,'!')
ans=
     good morning

strtok (string)
ans=
     good
```

3.1.11 字符串和数值的相互转换

MATLAB 中的字符串/数值转换函数可以将数值与字符串相互转换，既可以用这些函数来将数值转换成数字字符所组成的字符串，或转换成十六进制或二进制的字符串，反之亦同．例如：

```
x=1234;
y=int2str (x)
y=
   1234
size (y)
```

```
ans=
      1   4
```

上述例中用int2str 函数将整数 x 转换成 4 个字符的字符串．

可以利用nuim2str 函数来控制将数值转换成字符串的特定格式的字符串输出．在这个函数的第 2 个输入参数中，可以选择要输出数字的位数、例如：

```
p=num2str (pi, 9)
p=
   3.14159265
```

在 MATLAB 中，mat2str 函数可以将数组转换为字符串．例如．先创建一个 2 行 3 列的数组：

```
A= [1 2 3; 4 5 6]
A=
    1    2    3
    4    5    6
```

然后用 mat2str 来将 A 这个数组转换成字符串 B：

```
B=mat2str (A)
B=
     [1 2 3; 4 5 6]
```

3.2 符号矩阵的运算

在 MATLAB 的数值计算中，矩阵的加、减、乘和除等运算的操作指令都很直观简单．在符号计算中，情况就不同了，所有涉及符号计算的操作指令都要借助专用函数进行．

3.2.1 符号矩阵的创建

在数值计算过程中，所运作的变量都是被赋了值的数值变量．在符号计算的整个过程中，所运作的是符号变量．在这节中着重应用sym 指令创建符号矩阵的方法．

(1) sym 指令创建符号矩阵的直接方法

这是模仿MATLAB 数值矩阵的直接输入法设计的，矩阵元素可以是任何(不带等号的符号表达式)，矩阵中的各元素的长度可以不同，其各行之间用分号（;）隔开，其同行元素之间用逗号（,）分隔．例如：

```
T=sym (' [a, s, d; e, g, f]')
T=
     [a, s, d]
     [e, g, f]
```

(2) 创建符号矩阵的字符直接输入法

这是模仿 MATLAB 字符串矩阵的直接输入法设计的．需要注意，在应用这种方法时，符号矩阵的同一列元素字符串应具有同样的长度，因此，在较短字符串的前或后可用空格符填充．例如：

```
T=['[1/a,1+s,exp(x)]';'[e ,g ,x^2 ]']
T=
[1/a,1+s,exp(x)]
[e ,g ,x^2 ]
```

（3）把数值矩阵转化为符号矩阵

运用 sym 指令就可以把数值矩阵转化为符号矩阵．例如：

```
Mun=[2/3,sqrt(3)/3,0.333;2.5,1/0.7,log(3)]
Mun=
    0.6667    0.5774    0.3330
    2.5000    1.4286    1.0986

sym(Mun)
ans=
    [2/3,sqrt(1/3),333/1000]
    [5/2,10/7,4947709893870347*2^(-52)]
```

注意：上述运作过程中，不管数值矩阵 Mun 的元素原先是用分数还是浮点数表达，转化后的符号矩阵都将以最接近的精确有理形式给出．

另外，如果把这个指令与 MATLAB 中许多数值特殊矩阵生成指令（如 eye，ones，zeros，magic）配合使用，可以产生许多特殊的符号矩阵．

3.2.2 符号矩阵的加、减、乘和除运算

实现符号矩阵加（add）、减（subtract）、乘（multiply）运算的指令分别是：

symadd（A，B）　给出两个符号矩阵的和（A+B）；

symsub（A，B）　给出两个符号矩阵的差（A−B）；

symmul（A，B）　给出两个符号矩阵的乘积（A×B）．

说明：

1）symadd（A，−B）决不意味着 A+（−B）=A−B，因为（−B）对符号运算是非法的；

2）相乘规则是：符号表达式可以与符号矩阵相乘；两个维数相同的符号矩阵可以相乘．

例 3.5　求 $\frac{1}{(s+1)(s+3)}$ 与 $\begin{bmatrix} s+1 & s \\ 0 & s+4 \end{bmatrix}$ 的相乘．

```
G=symmul('1/(s+1)/(s+3)',sym('[s+1,s;0,s+4]'))
G=
   [1/(s+3),1/(s+1)/(s+3)*s]
   [0,1/(s+1)/(s+3)*(s+4)]
```

例 3.6　求 $\begin{bmatrix} s+1 & s \\ 0 & s+4 \end{bmatrix}$ 与 $\begin{bmatrix} \frac{1}{s(s+2)} & \frac{1}{(s+1)(s+2)} \end{bmatrix}^T$ 的相乘．

```
G=symmul(sym('[s+1,s;0,s+4]'),sym('[1/s/(s+2);1/(s+1)/(s+2)]'))
G=
```

```
[ (s+1)/s/(s+2)+s/(s+1)/(s+2)]
[(s+4)/(s+1)/(s+2)]
```

3.2.3 符号矩阵的逆和除运算

符号矩阵的求逆(inverse)指令和符号矩阵的除(divide)运算指令如下：

inv(B) 求 B^{-1}

symdiv(A,B) 计算 A/B，即 AB^{-1}

例 3.7 求$\begin{bmatrix} s+1 & s \\ 0 & s+4 \end{bmatrix}$的逆.

```
inv(sym('[s+1,s;0,s+4]'))
ans=
    [1/(s+1), -s/(s+1)/(s+4)]
    [0,1/(s+4)]
```

例 3.8 已知 $A=\begin{bmatrix} s+1 & s \\ 0 & s+4 \end{bmatrix}$，$B=\begin{bmatrix} s+2 & s \\ 1 & s+5 \end{bmatrix}$，求 A/B.

```
A=sym('[s+1,s;0,s+4]')
A=
   [s+1,s]
   [0, s+4]
B=sym('[s+2,s;1,s+5]')
B=
   [s+2,s]
   [1, s+5]
symdiv(A,B)
ans=
    [(5+5*s+s^2)/(6*s+10+s^2),s/(6*s+10+s^2)]
    [-(s+4)/(6*s+10+s^2), (s^2+6*s+8)/(6*s+10+s^2)]
```

3.2.4 符号矩阵的幂运算

与数值计算相比，符号计算对幂(power)运算所加的限制条件更多，否则所得结果就很难保证正确．具体如下：

sympower(S,p) 求幂运算指令 S^p，与 sym(A)^sym(B)相同

说明：若 S 为标量符号表达式，p 可为标量符号或数值表达式；若 S 为符号方阵，p 必须为整数．例如：

例 3.9 求符号方阵 $A=\begin{bmatrix} s+1 & s \\ 0 & s+4 \end{bmatrix}$的 2 次幂.

```
A=sym('[s+1,s;0,s+4]')
A=
   [s+1,s]
   [0, s+4]
```

```
sympow(A,2)
ans=
    [(s+1)^2, (s+1)*s+s*(s+4)]
    [0,(s+4)^2]

sym(A)^2
ans=
    [(s+1)^2, (s+1)*s+s*(s+4)]
    [0,(s+4)^2]
```

3.2.5 符号矩阵的综合运算指令

除了前面介绍的单种符号计算指令外，在符号数学工具箱中还有一个综合运算指令 symop. 提供此指令是出于两个考虑：第一是实际使用的需要；当多个符号矩阵或混有数值矩阵，要实现相互间的多种运算时，假如依靠单种运算指令去做，就显得缺乏效率. 第二是软件编制上的方便.

符号矩阵的综合运算指令：

symop(s1,s2,s3,...)　　　　符号矩阵的综合运算

其中 s1,s2,s3,... 分别是符号矩阵或数值矩阵或'+','-','*','/','^','(',')'中某算符.

例 3.10 综合运算指令与多次调用单种运算指令的比较.

```
A=sym('[a1,a2;a3,a4]');B=sym('[b1,b2;0,b4]');N=[1,2;3,4];
F=symop(A,'/','(',B,'+',N,')')

F=
[(-b4*a1-4*a1+3*a2)/(3*b2+2-b4*b1-b4-4*b1),-(b1*a2-
2*a1+a2-a1*b2)/(3*b2+2-b4*b1-b4-4*b1)]
[-(b4*a3+4*a3-3*a4)/(3*b2+2-b4*b1-b4-4*b1),(-b1*a4-
a4+2*a3+a3*b2)/(3*b2+2-b4*b1-b4-4*b1)]
FF=symdiv(A,symadd(B,sym(N)))  %由单种运算指令复合而成的输入指令

FF=
[(-b4*a1-4*a1+3*a2)/(3*b2+2-b4*b1-b4-4*b1), -(b1*a2-
2*a1+a2-a1*b2)/(3*b2+2-b4*b1-b4-4*b1)]
[-(b4*a3+4*a3-3*a4)/(3*b2+2-b4*b1-b4-4*b1), (-b1*a4-
a4+2*a3+a3*b2)/(3*b2+2-b4*b1-b4-4*b1)]
```

3.2.6 符号变量替换

在本章开头，已经介绍了如何利用变量替换指令把符号解中的自由参数替换成数字

符．本节还将更全面地介绍变量替换指令，实现这功能的指令：一个是 subs，它适用于单个符号矩阵、符号表达式、符号代数和微分方程；另一个是 symvars，它适用于符号表达式组、符号代数方程组，替代速度快．它们的具体使用格式如下：

subs(S,NEW)　　用新变量 NEW 替代 S 中的默认变量．

subs(S,NEW,OLD)　　用新变量 NEW 替代 S 中的指定变量 OLD.

例 3.11 符号矩阵的变量替换．

```
D=sym('[a * cos(b * x),a+b;exp(a * x),a]');
D1=subs(D,pi/3)
D1=
    [a * cos(1/3 * b * pi),a+b]
    [exp(1/3 * a * pi),a]

D2=subs(D,'2','a')
D2=
    [2 * cos(b * x),2+b]
    [exp(2 * x),2]
```

3.2.7 符号矩阵的分解

与数值计算一样，在符号计算中，符号矩阵分解也是不可缺少的．计算符号矩阵的行列式、转置、特征值分解、奇异值分解等指令如下：

determ(S)　　求 S 阵的行列式．

transpose(S)　　求 S 的符号转置矩阵．

colspace(S)　　给出 S 列空间的基．

[VE,E]=eigensys(S)　　VE 给出 S 的特征向量，E 给出 S 的特征值．

[VJ,J]=jordan(S)　　VJ 给出 S 的广义特征向量，J 给出相应的约当标准形．

singvals(A)　　给出一般符号阵 A 的奇异值．

[U,S,V]=singvals(A)　　给出 A 阵的奇异值分解三对组．

例 3.12 无重根阵的分解．

```
D=sym('[1,-3;2,2/3]');
[VE,E]=eigensys(D)
VE=
    [1,1]
    [ 1/18-1/18 * i * 215^(1/2), 1/18+1/18 * i * 215^(1/2)]

E=
    [5/6+1/6 * i * 215^(1/2),0]
    [0, 5/6-1/6 * i * 215^(1/2)]
```

例 3.13 有重根阵的分解．

```
C=sym('[1,1,2;0,1,3;0,0,2]');
```

```
[VJ,J]=jordan(C)
VJ=
    [5,  -5,-5]
    [3,   0,-5]
    [1,   0,   0]

J=
    [2,   0,   0]
    [0,   1,   1]
    [0,   0,   1]
```

3.2.8 符号微积分

在这一小节中，介绍符号和、符号导数和符号积分的求取，具体指令如下：

sum(S)　　当 S 是符号矢量时，该指令对所有 S 元素求和；当 S 是符号矩阵时，该指令对指定的列进行求和．

diff(S,'v')　　计算符号矩阵 S 对指定变量 v 的一阶导数．

diff(S,'v',n)　　计算符号矩阵 S 对指定变量 v 的 n 阶导数．

int(S,'v')　　计算符号矩阵 S 对指定变量 v 的不定积分．

int(S,'v',a,b)　　计算符号矩阵 S 对指定变量 v 在(a,b)区间内的定积分．

例 3.14 应用 sum 指令对符号向量和符号矩阵进行计算．

```
sum(sym('[log(a)^k,X^k,2,1]'))
ans=
    log(a)^k+X^k+3

sum(sym('[k,a-k;2,b-k]'))
ans=
    [k+2, a-2*k+b]
```

例 3.15 求$\frac{\partial}{\partial x\partial y}\begin{bmatrix} x\sin(y) & x^n+y \\ \frac{1}{xy} & e^{ixy} \end{bmatrix}$.

```
S=sym('[x*sin(y),x^n+y;1/x/y,exp(i*x*y)]');
dsdxdy=diff(diff(S,'x'),'y')
dsdxdy=
        [cos(y),0]
        [1/x^2/y^2, i*exp(i*x*y)-y*x*exp(i*x*y)]
```

例 3.16 求$\iint xe^{-xy}dxdy$.

```
int(int('x*exp(-x*y)','x'),'y')
ans=
1/y*exp(-x*y)
```

3.2.9 符号代数方程的求解

在这节里，只介绍线性方程组的符号解和一般方程组的符号解.

(1)线性方程组的符号解

考虑 A 阵至少行满秩的线性方程组 A * X=B 的解，可采用下面的指令：

X=linsolve(A,B)　　仅仅给出特解.

例 3.17 求线性方程组的解.

```
A=sym('[1,1/2,1/3;3,1,1;1,2,1]');
B=sym('[1,2;1/3,1;1,1/7]');
[x]=linsolve(A,B)
x=
   [7/3,38/7]
   [16/3,10]
   [-12, -177/7]
```

实际上，linsolve(A,B)与 sym(A)\sym(B)等价，如：

```
A='[1,1/2,1/3;3,1,1;1,2,1]';
B='[1,2;1/3,1;1,1/7]';
sym(A)\sym(B)
ans=
     [7/3,38/7]
     [16/3,10]
     [-12, -177/7]
```

(2)一般方程组的符号解

MATLAB 提供的 solve 指令能解一般代数方程，包括线性、非线性和超越方程，其具体使用格式如下：

solve('expr1,expr2,...,exprN','var1,var2,varN')对 N 个方程表示式(expr)指定变量(var)求解.

solve('expr1,expr2,...,exprN')对 N 个方程表示式(expr)的默认变量求解.

例 3.18 求 $x^2-x-6=0$ 的解.

```
solve('x^2-x-6')
ans=
     [ -2 ]
     [  3 ]
```

例 3.19 求 $\begin{cases} x+y-1=0 \\ x-y-2=0 \end{cases}$ 的解.

```
[X,Y]=solve('x+y-1, x-y-2')
X=
   3/2
```

```
Y=
  -1/2
```

例 3.20　求$\begin{cases} tx+zy-1=0 \\ 2tx-3zy+2=0 \end{cases}$的解．

```
[t,z]=solve('t*x+z*y-1, 2*t*x-3*z*y+2','t,z')
t=
  1/5/x


z=
  4/5/y
```

3.2.10　符号微分方程的求解

在 MATLAB 中，常微分方程的求解指令为 dsolve，它的使用格式为

[y1,y2,…]=dsolve('eqn1,eqn2,…')

说明

1）输入量包括三个部分内容：微分方程、初始条件和指定独立变量，其中微分方程是必不可少的，而后面两项可视需要而定．

2）输出量可有可无．当有输出量时，MATLAB 工作内存中将在 y1，y2，... 定义的输出量中保存计算结果．

3）微分方程的记述规定：当"y"是因变量时，用"Dny"表示 y 的 n 阶导数，例如：

Dny　　表示形如$\frac{d^n y}{dx^n}$的导数．

例 3.21　求 $\frac{dx}{dt}=-ax$ 的解．

```
dsolve('Dx=-a*x')
ans=
    exp(-a*t)*C1
```

例 3.22　求 $\frac{dx}{dt}=-ax$ 的解，初始条件为 $x(0)=1$.

```
x=dsolve('Dx=-a*x','x(0)=1','s')
x=
  exp(-a*s)
```

例 3.22　求$(\frac{dy}{dt})^2+y^2=1$ 的解，初始条件为 y(0)=1.

```
y=dsolve('(Dy)^2 + y^2=1','y(0)=0')
y=
  [-sin(t)]
  [sin(t)]
```

例 3.24　求$\begin{cases} \frac{df}{dt}=f+g \\ \frac{dg}{dt}=-f+g \end{cases}$的解，初始条件为$\begin{cases} f(0)=1 \\ g(0)=2 \end{cases}$

```
[f,g]=dsolve('Df=f + g,Dg=-f + g','f(0)=1,g(0)=2')
f=
   exp(t)*cos(t)+2*exp(t)*sin(t)


g=
   -exp(t)*sin(t)+2*exp(t)*cos(t)
```

例 3.25 求解三阶微分方程$\frac{d^3y}{dt^3}=-y$,初始条件为 $y(0)=1,\frac{dy^2(0)}{dt^2}=0,\frac{dy(0)}{dt}=0$.

```
y=dsolve('D3y=-y','y(0)=1,Dy(0)=0,D2y(0)=0','t')
y=
   (1/3+2/3*exp(1/2*t)*cos(1/2*3^(1/2)*t)*exp(t))/exp(t)
```

第四章　绘　　图

仿真数据的图形化是广大科技工作者研究问题所不可缺少的手段．MATLAB 不仅在数值计算方面无与伦比，而且在数据图形化方面也有它的优势．

1）MATLAB 能够给数据以二维、三维乃至四维的图形表现．通过对图形线型、立面、色彩、渲染、光线、视角等品性的处理，可把计算数据的特征表现得淋漓尽致．

2）MATLAB 绘图功能建立在一组“图形对象（graphics objects）”基础之上，其核心是“图形句柄（graphics handle）”操作．

3）MATLAB 有两层绘图指令：一组是直接对句柄进行操作的底层（low-level）绘图指令；另一组是在底层指令基础上建立起来的高层（high-level）绘图指令．

高层绘图指令简单明了，比较容易掌握，底层绘图指令具有很强的控制和表现数据图形的能力，比较灵活多变．本章内容按“先易后难”的原则安排，使读者尽快地掌握 MATLAB 绘图功能．

4.1　二维绘图

4.1.1　plot

plot 是最基本的二维绘图命令，在二维绘图中，只要输入 plot（a，b）这个命令，就可以画出一个以 a 为 x 轴、b 为 y 轴的图形．

plot 的基本调用格式有 3 种：

（1）plot（x）

若 x 为向量，则以 x 元素值为纵坐标，以相应元素下标为横坐标值，绘制连线图．若 x 为实数阵，则按列绘制每列元素值相对其下标的连线图，图中曲线等于 x 阵的列数；若 x 为复数阵，则分别以 x 实部阵和虚部阵的对应列元素为横纵坐标绘制多条连线图．

（2）plot（x，y）

若 x，y 是同维向量，则绘制以 x，y 元素为横纵坐标的连线图．若 x 是向量，y 是一个与 x 等维的矩阵，则绘制出多根不同色彩的连线图，连线根数等于 y 阵的另一个维数．若 x，y 是等维矩阵，则以 x，y 对应列元素为横纵坐标分别绘制曲线，曲线的根数等于矩阵的行数．

（3）plot（x1，y1，x2，y2，…）

每个二元对 x-y 的作用与 plot（x，y）相同，不同二元对之间没有约束关系．

例 4.1　输入如图 4.1 所示的程序：

```
x=[1,3,5,4,6,2,0,1,9];
plot(x);
```

图 4.1　范例 4.1

就可以在屏幕上看到一个单向输入格式所画的连线，如图 4.2 所示．

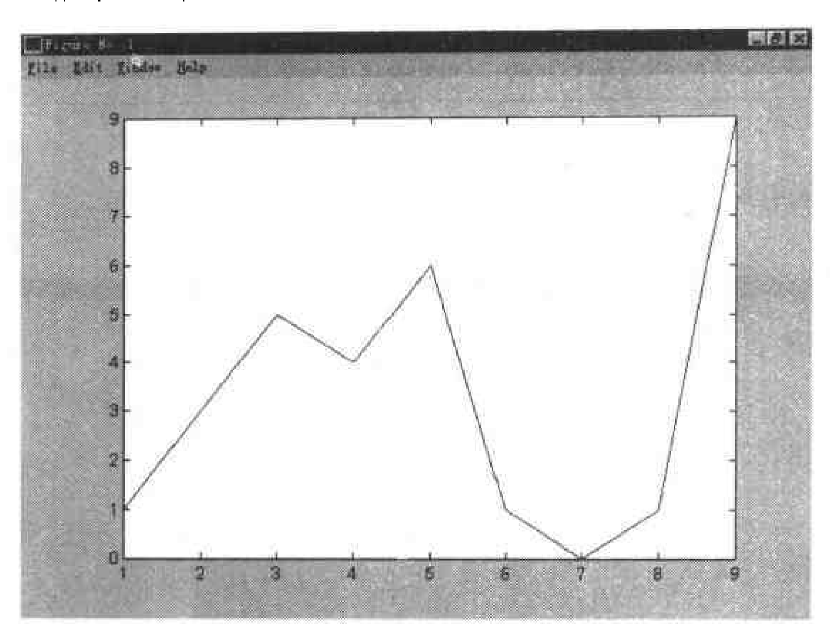


图 4.2 范例 4.1 的运行结果

例 4.2 输入如图 4.3 所示的程序．

```
x=0:pi/100:2*pi;
plot(x,sin(x));
```

图 4.3 范例 4.2

就可以在屏幕上看到一个单向输入格式所画的曲线，如图 4.4 所示．

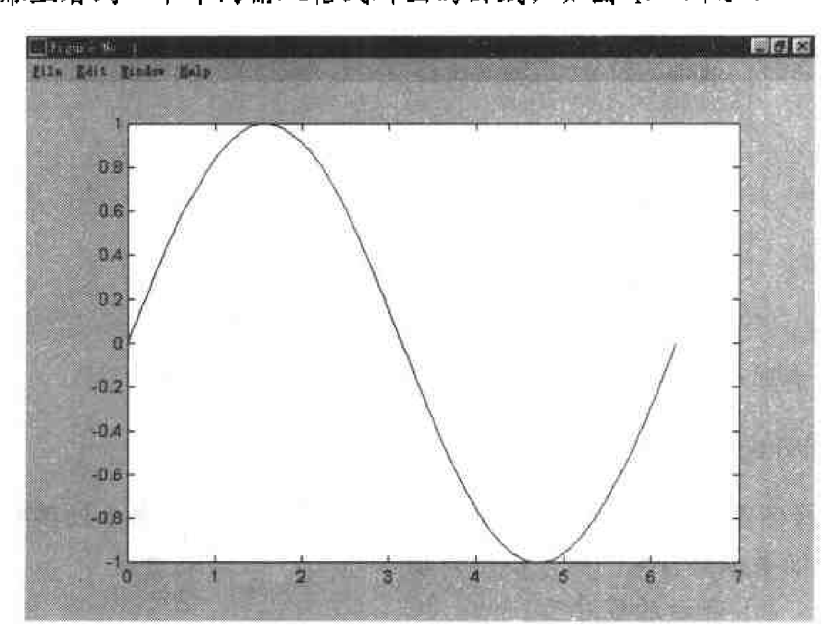


图 4.4 范例 4.2 的运行结果

例 4.3 输入如图 4.5 所示的程序．

```
x=0:pi/100:2*pi;
plot(x,sin(x),x,cos(x),x,cos(x+0.5));
```

图 4.5　范例 4.3

就可以在屏幕上看到三个单向输入格式所画的曲线，如图 4.6 所示．

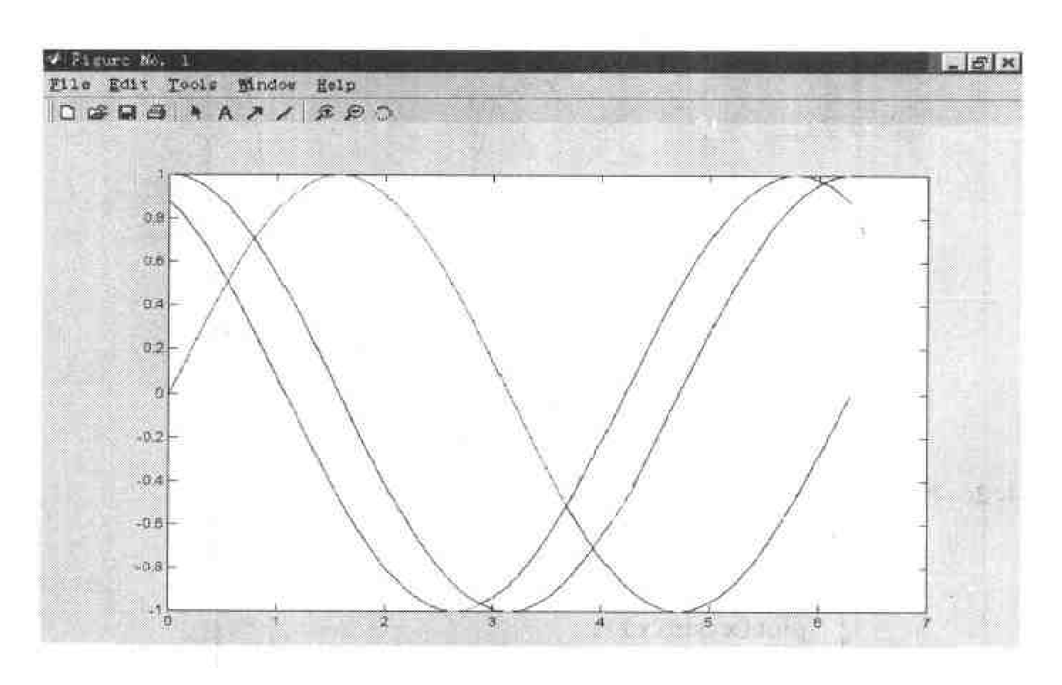


图 4.6　范例 4.3 的运行结果

例 4.4　输入 Euler 公式($e^{ix}=\cos(x)+i\sin(x)$)，程序如图 4.7 所示．

```
x=0:pi/100:22*pi;
y=exp(i*x);
plot(y);
axis('square');
```

图 4.7　范例 4.4

就可以在屏幕上看到一个单向输入格式所画的曲线，如图 4.8 所示．

4.1.2　figure 和 subplot

figure 是选择图像的命令．有时候我们可能画出几张图，屏幕上所能看到的只有最近打开的那张．如果想再看第一张，就可以输入 figure (1) 命令．

例 4.5　输入 4.9 所示的程序后

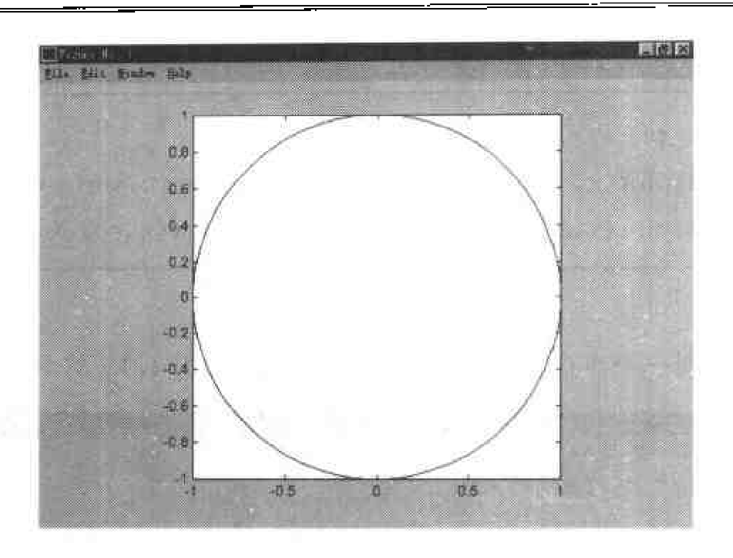


图 4.8　范例 4.4 的运行结果

```
x=0:pi/100:2*pi;
figure(1);
plot(x,sin(x));
figure(2);
plot(x,cos(x));
```

图 4.9　范例 4.5

出现如图 4.10 所示的两个窗口．可以用 figure（1）与 figure（2）这两个命令来切换显示图 4.10 这两个窗口．

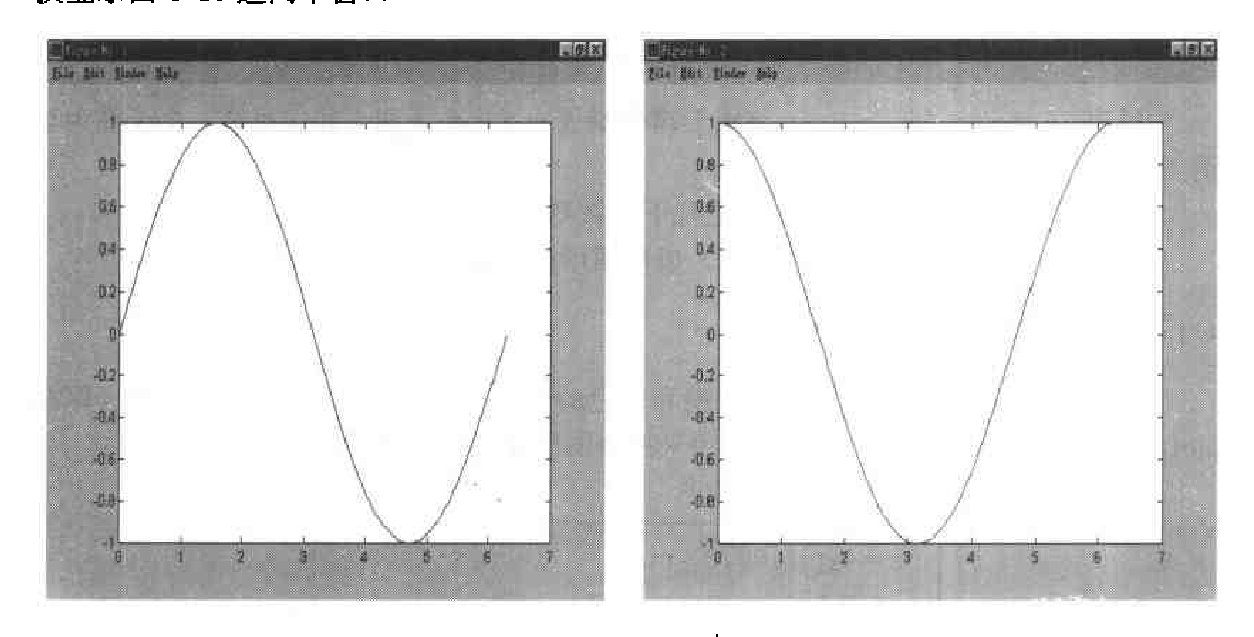


图 4.10　范例 4.5 的运行结果

subplot 是图形显示时分割窗口的命令．即这个命令可以实现在同一窗口里显示多个图形．

例 4.6 输入如图 4.11 所示的程序.

```
x=0:pi/100:2*pi;
subplot(2,2,1);plot(x,sin(x));subplot(2,2,2);plot(x,sin(x*1.5));
subplot(2,2,3);plot(x,cos(x));subplot(2,2,4);plot(x,cos(x*1.5));
```

图 4.11 范例 4.6

就可以在屏幕上看到一个单向输入格式所画的曲线，如图 4.12 所示.

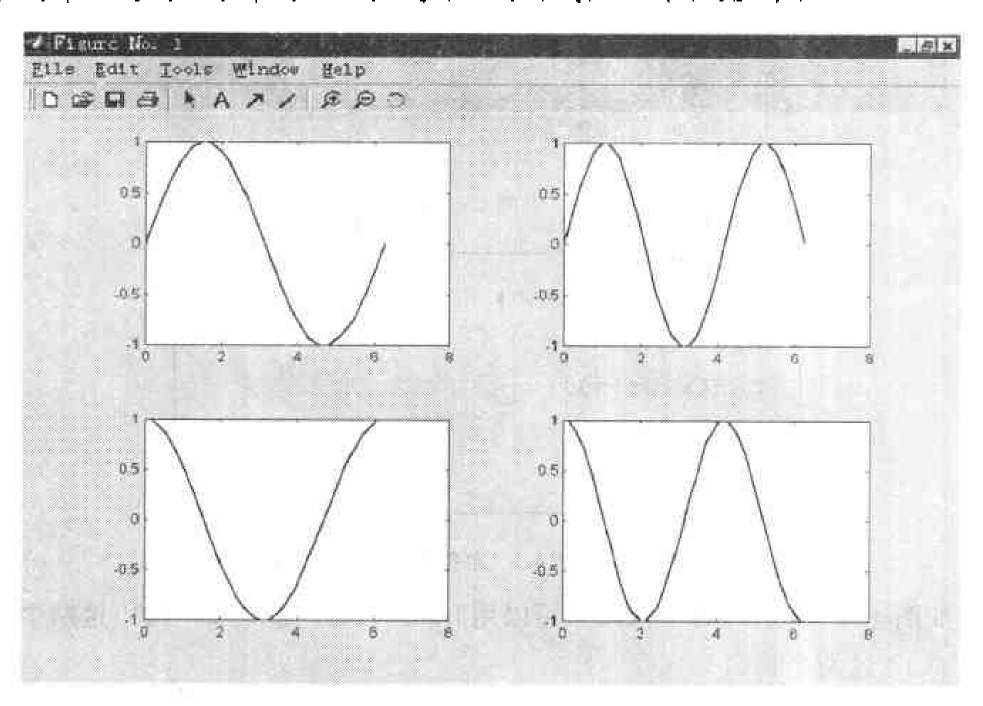


图 4.12 范例 4.6 的运行结果

说明：subplot (m, n, p) 将图形窗分割成 m×n 个子图，并选择第 p 个子图为当前图形.

注意：1) 子窗口的序号按行由上往下，按列由左向右编号.

2) 如果不用指令 clf 清除，以后图形将被绘制在子图形窗口中.

4.1.3 绘图指令的开关控制

plot 指令还提供一组控制曲线线型和颜色的开关. 具体用法如图 4.13 所示（即在 plot 命令中再多加一个或几个参数，用来控制图形颜色和线条样式）.

例 4.7 输入如图 4.11 所示的程序.

```
x=0:pi/100:2*pi;
y1=sin(x);y2=cos(x);
plot(x,y1,'r-',x,y2,'b--');
```

图 4.13 范例 4.7

就可以在屏幕上看到一个单向输入格式所画的曲线，如图 4.14 所示.

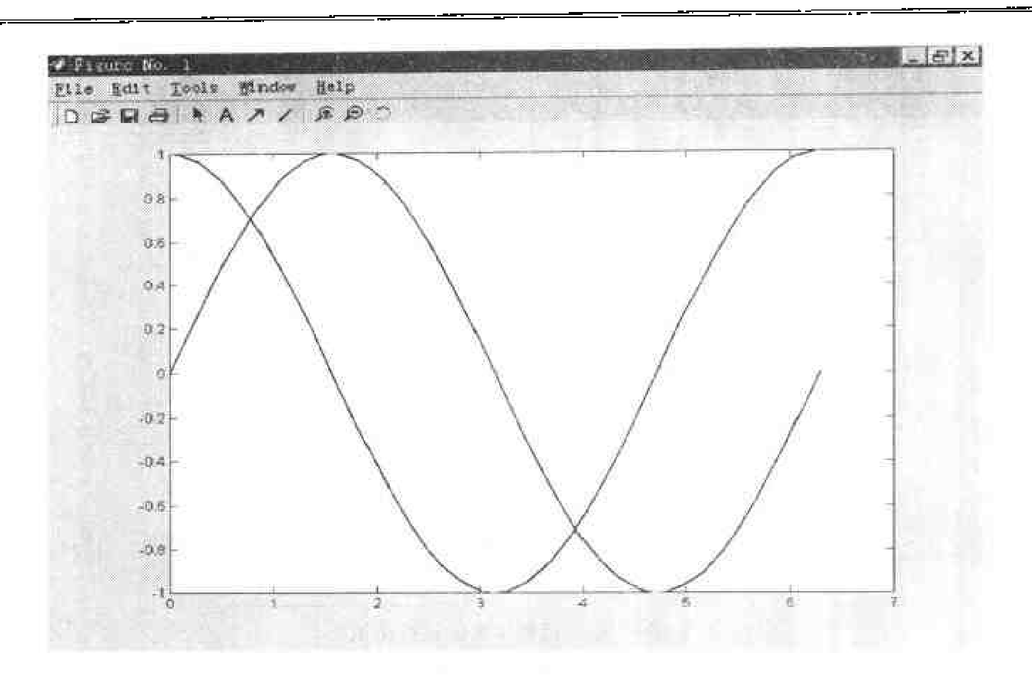


图 4.14　范例 4.7 的运行结果

指令中的参数含义如表 4.1 所示.

表 4.1　plot 附加参数的含义

参　数	含　义	参　数	含　义	参　数	含　义
Y	黄色	.	点	-	实线
M	紫色	O	圆	:	虚线
C	青色	X	打叉	-.	点划线
R	红色	+	加号	—	破折线
G	绿色	*	星号	<	向左三角形
B	蓝色	S	正方形	>	向右三角形
W	白色	D	菱形	p	五角星形
K	黑色	V	向下的三角形	h	六角星形
^	向上三角形				

hold on 和 hold off 可以在原来的图上加画其他线条或文字，而不会把原来的图覆盖掉. 这可以从下面例子体会到.

例 4.8　输入如图 4.15 所示的命令.

```
x=0:pi/100:2*pi;
plot(x,sin(x));
```

图 4.15　范例 4.8

此时会看到图 4.16 所示的图形. 然后改写如图 4.15 所示的程序为如图 4.17 所示，就可以在屏幕上看到一个单向输入格式所画的曲线，如图 4.18 所示.

例 4.9　输入如图 4.17 所示的程序.

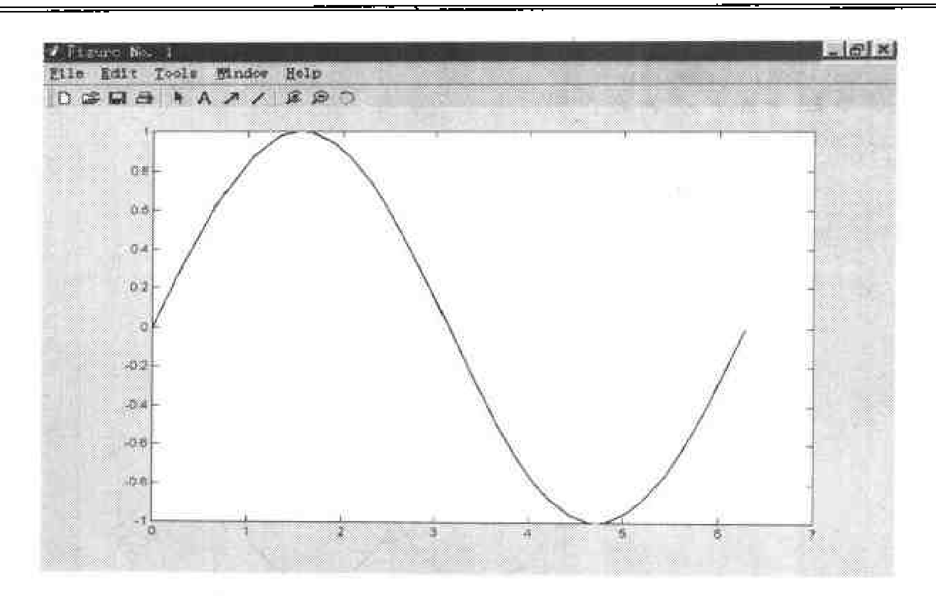


图 4.16　范例 4.8 的运行结果

```
x=0:pi/100:2*pi;
plot(x,sin(x));
hold on;
plot(x,sin(x+0.5),'r--')
```

图 4.17　范例 4.9

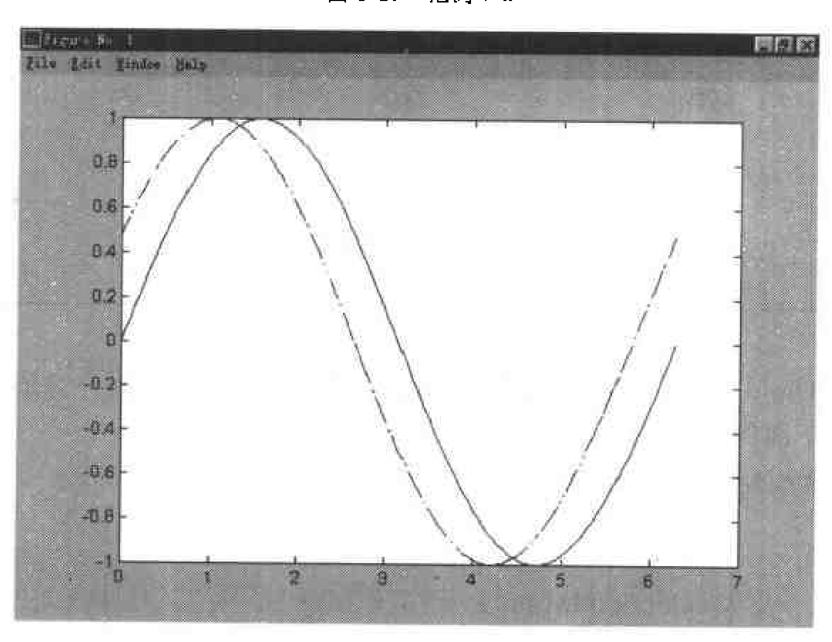


图 4.18　范例 4.9 的运行结果

如果想取消 hold on，就得用 hold off 来实现.

Grid on 这个命令可以在图形上画出坐标网络线，请看图 4.19.

例 4.10　输入如图 4.19 所示的程序.

```
x=0:pi/100:2*pi;
plot(x,sin(x));
grid on;
```

图4.19 范例4.10

其结果如图4.20所示.

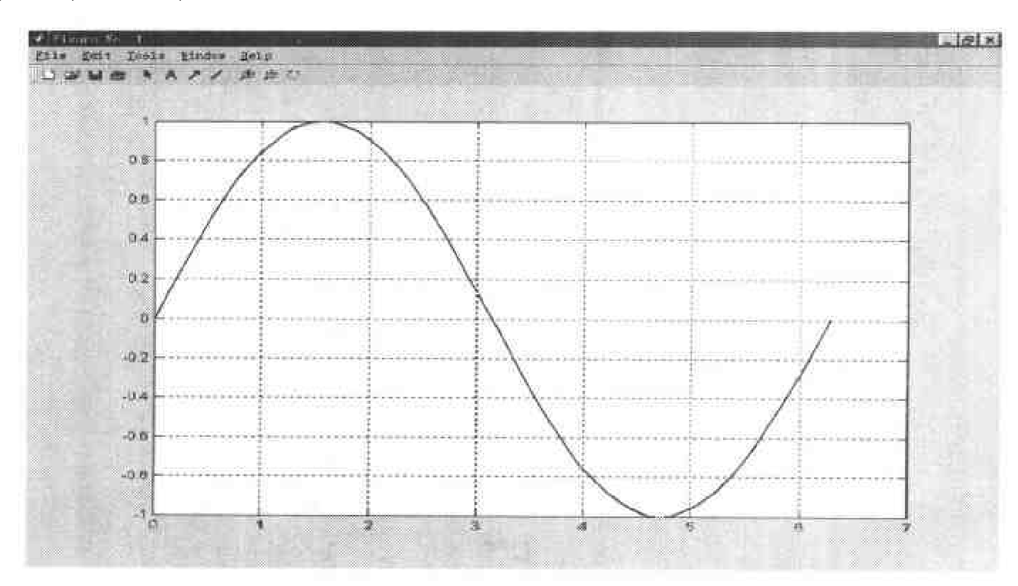


图4.20 范例4.10的运行结果

4.1.4 标题与坐标轴的操作

如果想在图形上加些记号,可用title指令,或者在图形中任意特定位置上做标识,可用text(x,y,'想加的标号'),或者在x轴与y轴上加标号,可用xlabel与ylabel.

例4.11 如图4.21所示.

```
x=0:pi/20:2*pi;
plot(x,sin(x));\=title('sin wave');
xlabel('x Value');
ylabel('sin(x)');
text(3*pi/4,sin(3*pi/4),'\leftarrowsin(x)=0.707');
text(5*pi/4,sin(5*pi/4),'sin(x)=-0.707\rightarrow','
                                    HorizontalAlignment','right');
```

图4.21 范例4.11

其结果如图4.22所示.

注意:text的默认方式是从插入点的右边开始写文字,所以如果要实现sin(x)=-0.707的效果,就要加上'HorizontalAlignment'和'right'.

在应用中,指数或对数的图形会遇到在log为坐标轴表示的情况.MATLAB提供了三个命令以实现此功能:loglog(x-y轴半对数图)、semilogx(x轴半对数图)和semilogy(y轴半对数图).

以semilogx为例,如图4.23所示.

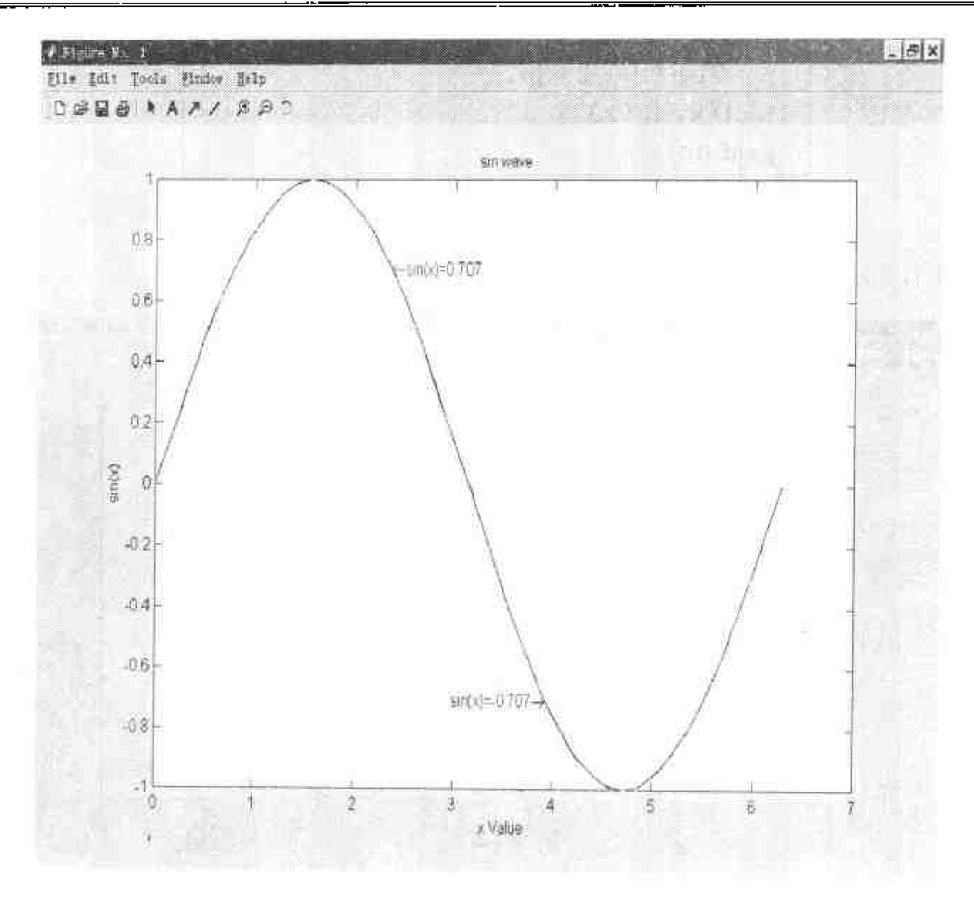


图 4.22　范例 4.11 的运行结果

例 4.12　输入如图 4.23 所示的命令：

```
x=0:1:1000;
semilogx(x,x);
```

图 4.23　范例 4.12

其结果如图 4.24 所示．

MATLAB 提供 axis 命令，用来设定坐标轴的范围，axis 的用法为 axis([xmin xmax ymin ymax])，如果有无穷大的情况发生，可用"inf"及"—inf"来表示．

例 4.13　输入如图 4.25 所示的程序，其结果如图 4.26 所示．

考虑到应用领域的图形是多种多样的，统一坐标模式不可能总是最有效地表现出所绘图形的特征．所以，MATLAB 设计了下列操纵坐标性质的 axis 指令：

axis ('ij')　以矩阵"(ij)"坐标轴表现图形．

axis ('xy')　使坐标轴返回到缺省状态的笛卡儿坐标系．

axis ('off ')　使坐标轴消隐．

axis ('on')　返回坐标轴显现状态．

axis ('equal')　使各坐标轴刻度增量相同．

axis ('square')　使各坐标轴长度相同（但刻度增量未必相同）．

axis ('normal')　使坐标轴恢复正常状态．

axis ('auto')　返回坐标轴的缺省状态．

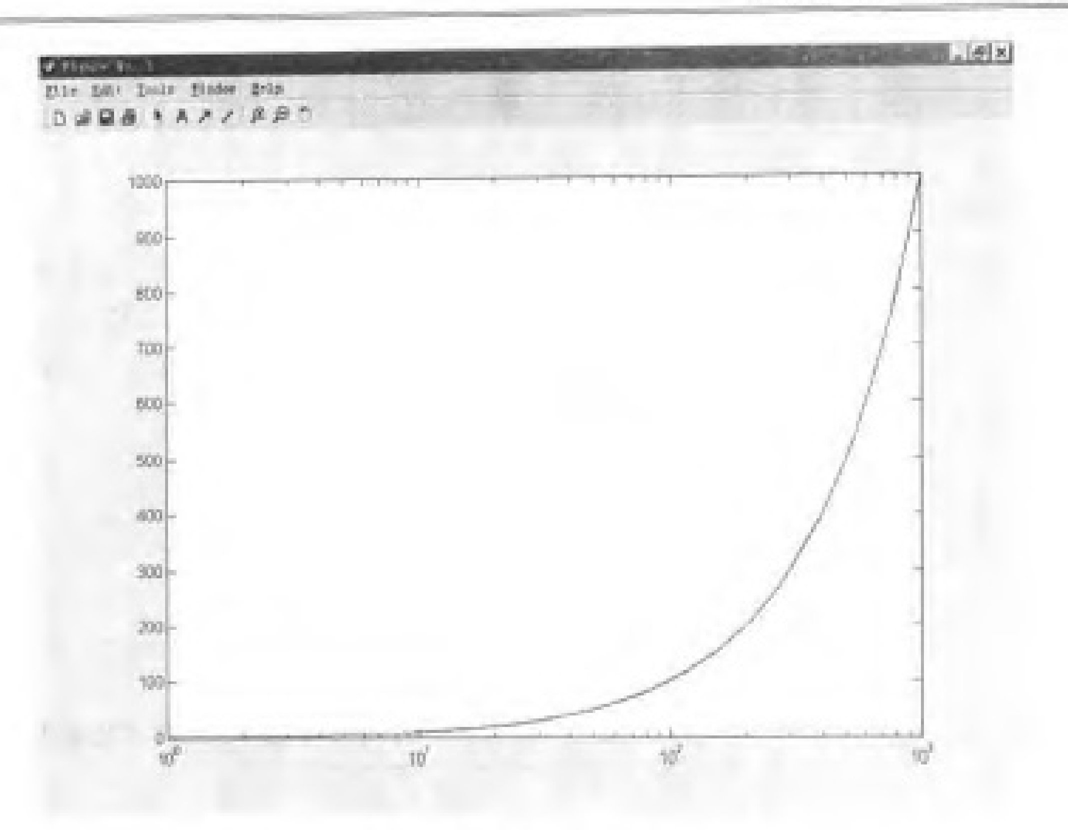


图 4.24　范例 4.12 的运行结果

```
x=0:pi/100:2*pi;
plot(x,sin(x));
axis([0  2*pi  -1  1])
```

图 4.25 范例 4.13

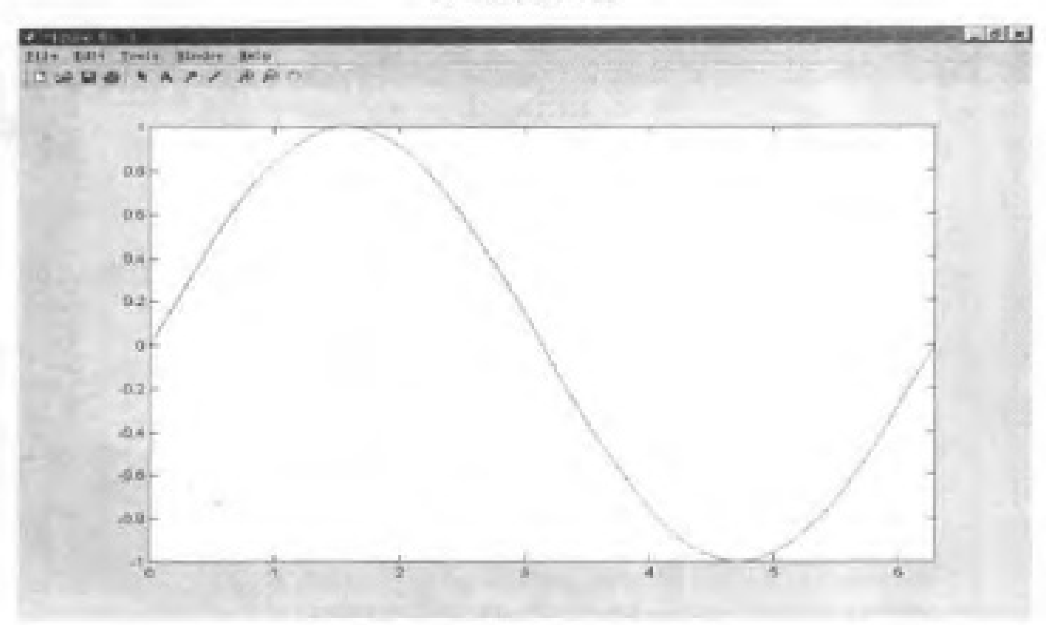


图 4.26　范例 4.13 的运行结果

4.2　三维绘图

4.2.1　mesh

mesh 是最基本的三维绘图命令，它的三维网线图的形成原理是：对 x-y 平面某一指定矩形范围采用与坐标轴平行的直线将其分格：计算矩形网格点上的函数值 $z=(x,y)$，得到 (x, y, z) 三维空间内的数据点；将这些数据点分别用处于 x-z 或其平行面内的曲线和处于 y-z 或其平行面内的曲线连接，即得 mesh 绘制出的三维图．

mesh 一般有两种最基本的调用格式：

(1) mesh (z)

以 z 矩阵元素值及其下标为数据点，绘制网线图．

(2) mesh (x，y，z)

若 $x\in R^n$，$y\in R^m$ 那么必须要求 $z\in R^{n\times m}$，网格数据点三维坐标为 x，y，z.

若 x，y，z 皆为 $R^{n\times m}$，则数据点的坐标分别取自这三个阵．

例 4.14　输入如图 4.27 所示的程序，运行后就可得到如图 4.28 所示的结果．

```
z=peaks(50);
h=mesh(z)
```

图 4.27　范例 4.14

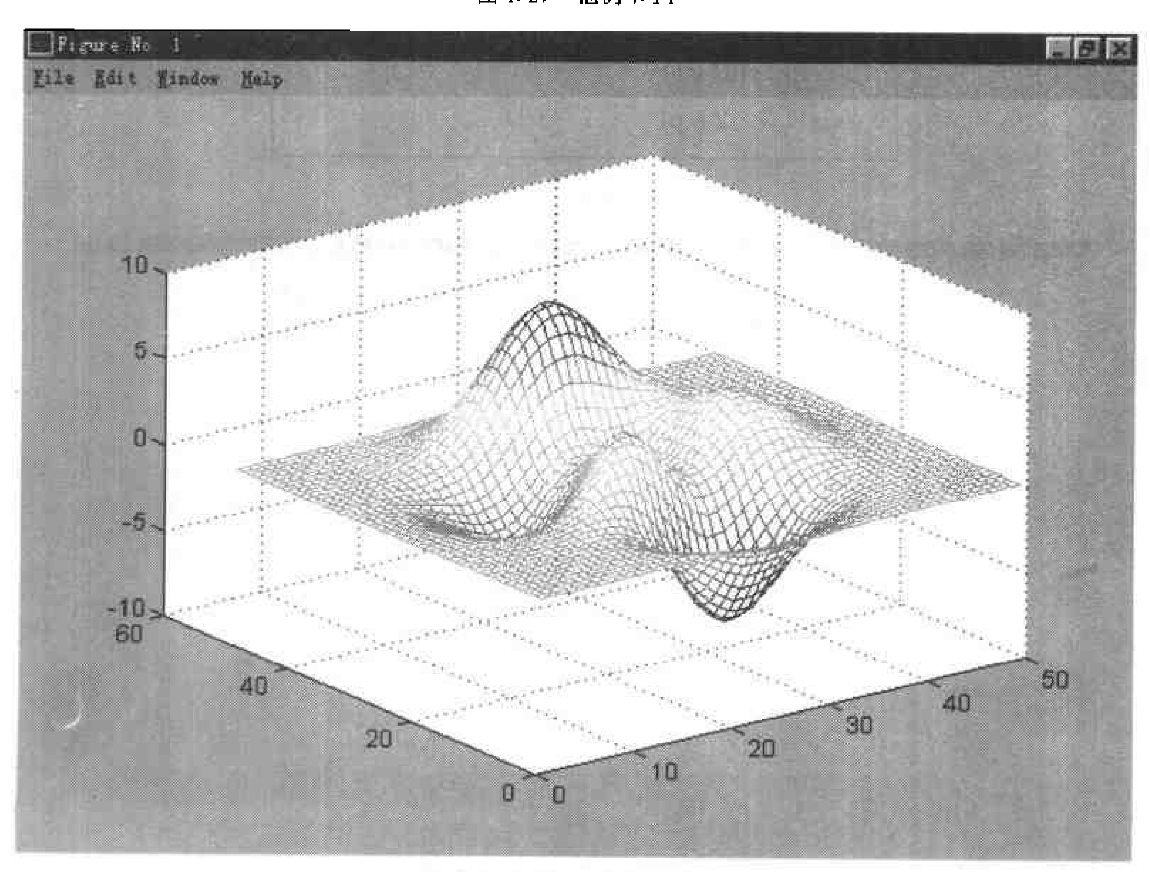


图 4.28　范例 4.14 的运行结果

另外，还有面状三维绘图函数 surf，仅就上例，可以把 mesh 换成 surf，如图 4.29 和图 4.30 所示．

例 4.15　输入如图 4.29 所示的程序．

```
z=peaks(50);
h=surf(z)
```

图 4.29　范例 4.15

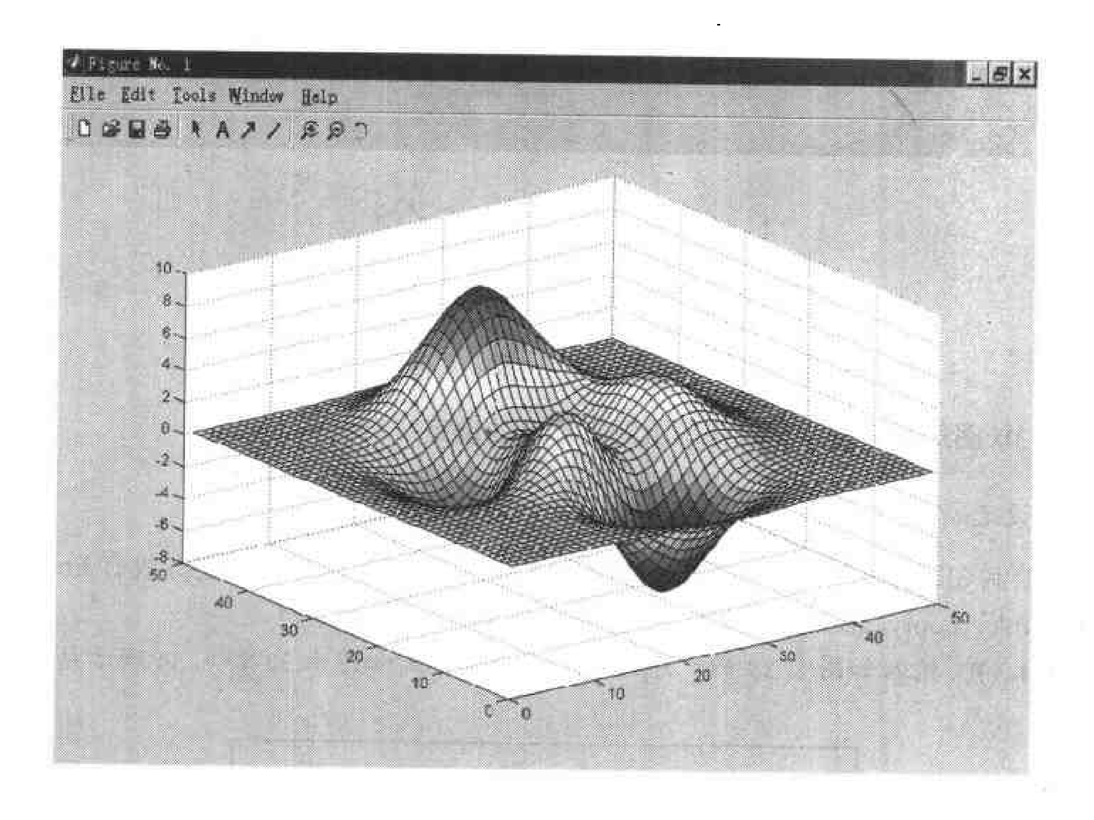


图 4.30　范例 4.15 的运行结果

例 4.16　可以把 mesh()，surf()和线性 3D 图形做个比较，绘制 $z=x^2+y^2$ 的三维网络图形，如图 4.31 和图 4.32 所示．

```
clear;
[x,y]=meshgrid([-4:0.1:4]);
z=peaks(x,y);
plot3(x,y,z)
```

图 4.31　范例 4.16

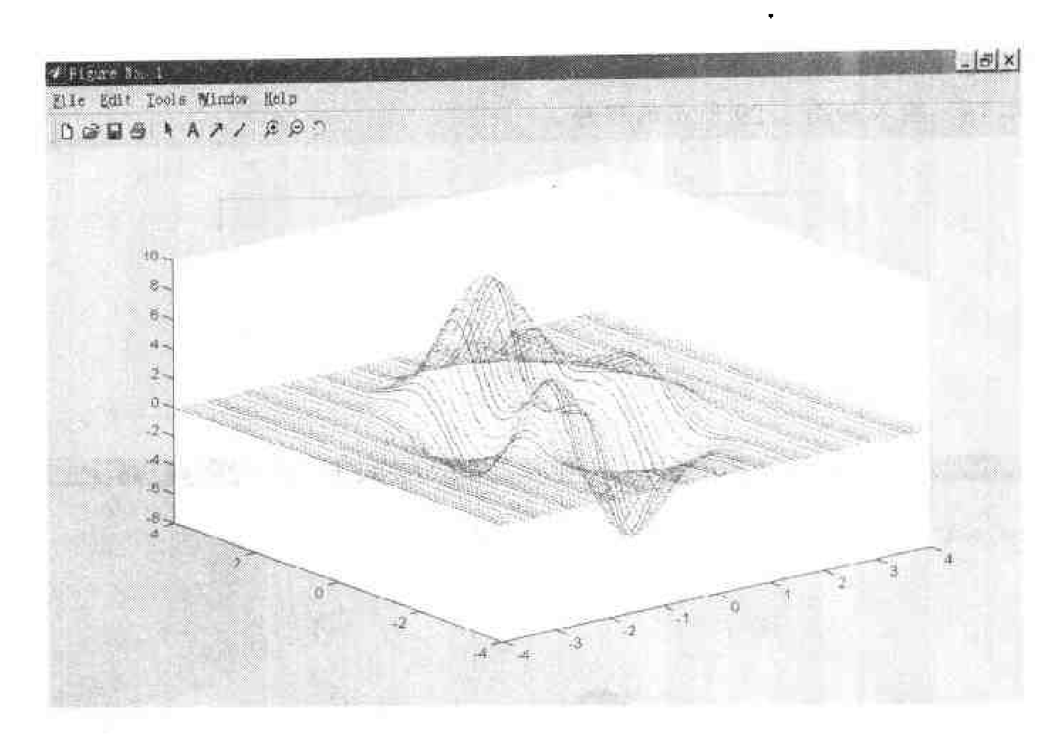


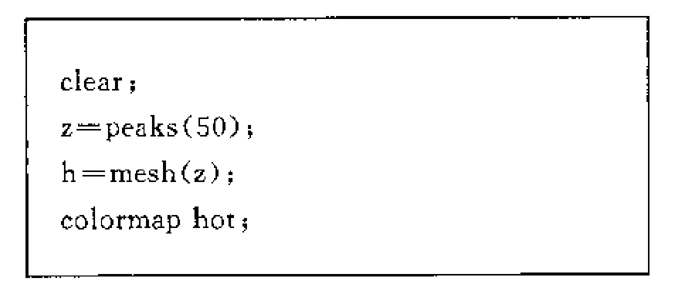
图 4. 32　范例 4. 16 的运行结果

4. 2. 2　3D 图形的颜色、光线来源及图上标点的设定

1. 设定颜色

在绘制 3D 图中，设计者可以选择图中的色彩分布，其颜色设定参数有 hot，hsv，gray，pink，cool，bone，copper. 具体应用可从例 4. 17 中体会.

例 4. 17　输入如图 4. 33 的程序，运行后就可得到如图 4. 33 和 4. 34 所示结果.



```
clear;
z=peaks(50);
h=mesh(z);
colormap hot;
```

图 4. 33　范例 4. 17

2. 设定光线来源

light 这个命令可以设定光源的方向，如图 4. 35 和图 4. 36 所示.

例 4. 18　输入如图 4. 35 所示的程序，运行后就可得到如图 4. 36 所示结果.

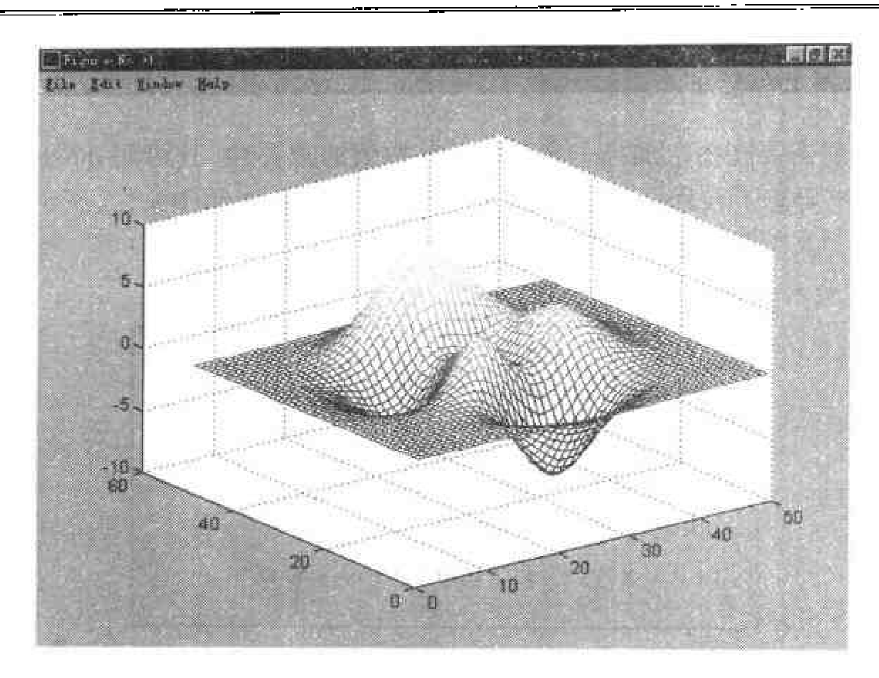


图 4.34 范例 4.17 的运行结果

```
clear;
z=peaks(50);
h=mesh(z);
light('position',[0 0 5]);
```

图 4.35 范例 4.18

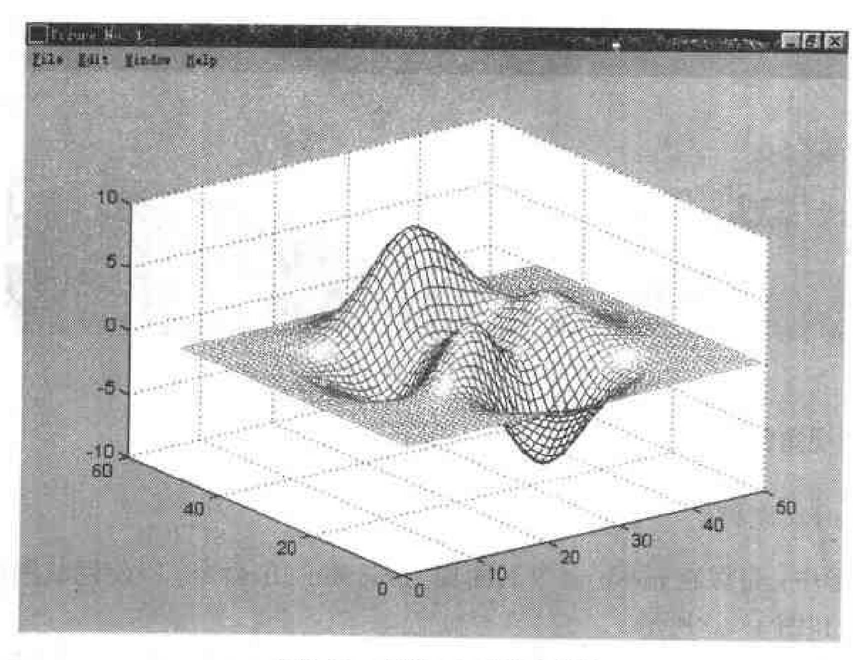


图 4.36 范例 4.18 的运行结果

3. 在三维图上加标志

有时候我们需要在图上加标志来显示某些数值的重要性，这可用mesh 与plot3 组合来实现，切记不要忘了在程序中加hold on 命令，如图 4.37 和图 4.38 所示，即在网线交叉处都会打上“×”的标志.

例 4.19 输入如图 4.37 所示的程序.

```
[x,y]=meshgrid([-3:0.2:3]);
z=peaks(x,y);
mesh(x,y,z);
hold on
plot3(x,y,z,'x','markersize',3);
```

图 4.37 范例 4.19

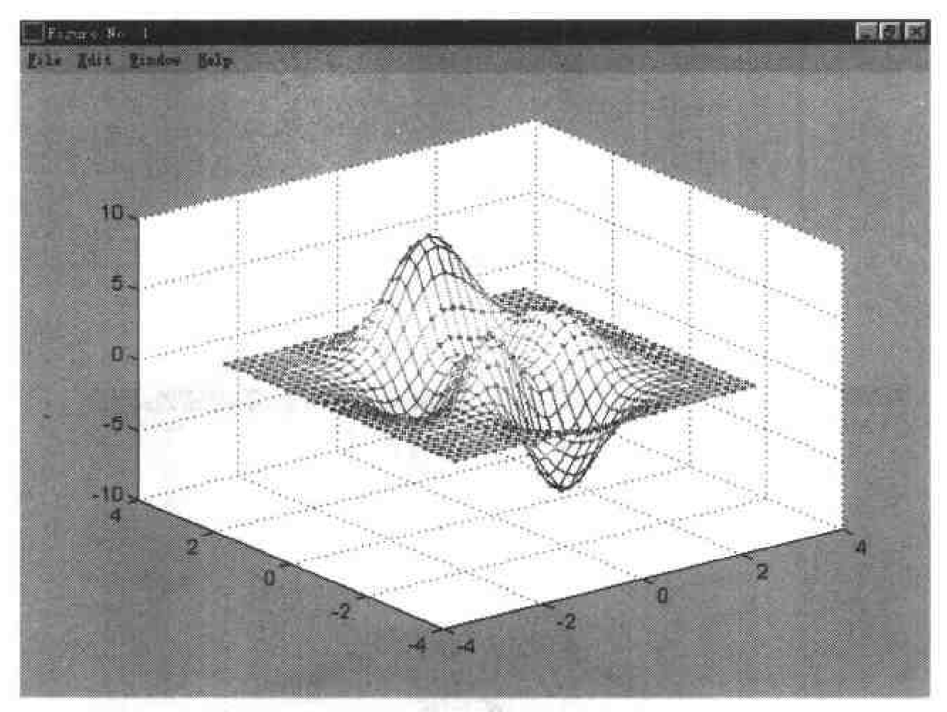


图 4.38 范例 4.19 的运行结果

4.2.3 透视与视角的设置

1. hidden off (透视)

在三维图形中，可以在mesh 命令后面加上 hidden off 命令，以使网状图产生透视效果，如图 4.39 和图 4.40 所示.

例 4.20 输入如图 4.39 所示的程序，运行后就可以得到如图 4.40 所示的结果.

```
z=peaks(x,y);
mesh(z);
hidden off
```

图4.39　范例4.20

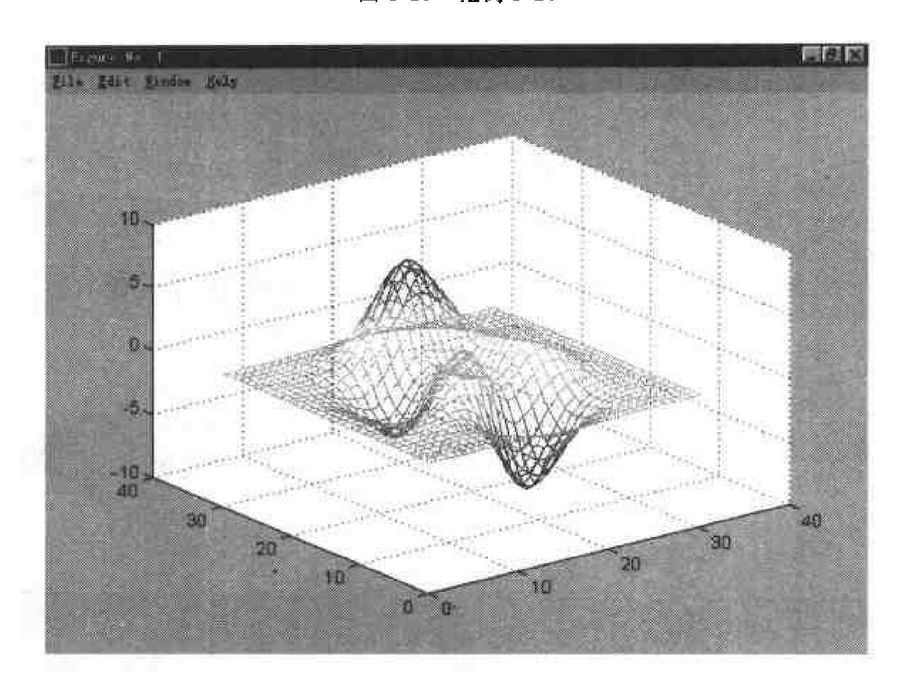


图4.40　范例4.20的运行结果

2. 设置三维图形的视角

三维图形的视角设置可通过view（观察函数）命令来实现，观察格式：view（az，el），其中az是方位角，el是俯视角，它们的单位均为度（进行二维观察时缺省值为az=0，el=90；进行三维观察时缺省值为az=37.5，el=30）.

例4.21　输入如图4.41（peaks函数的四种不同视图设置程序），可得如图4.42所示的结果.

```
z=peaks(40);
subplot(2,2,1);mesh(z);view(-37.5,30);title('方位角=-37.5,俯仰角=30');
subplot(2,2,2);mesh(z);view(-7,80);title('方位角=-7,俯仰角=80');
subplot(2,2,3);mesh(z);view(-90,0);title('方位角=-90,俯仰角=0');
subplot(2,2,4);mesh(z);view(-7,10);title('方位角=-7,俯仰角=10');
```

图4.41　范例4.21

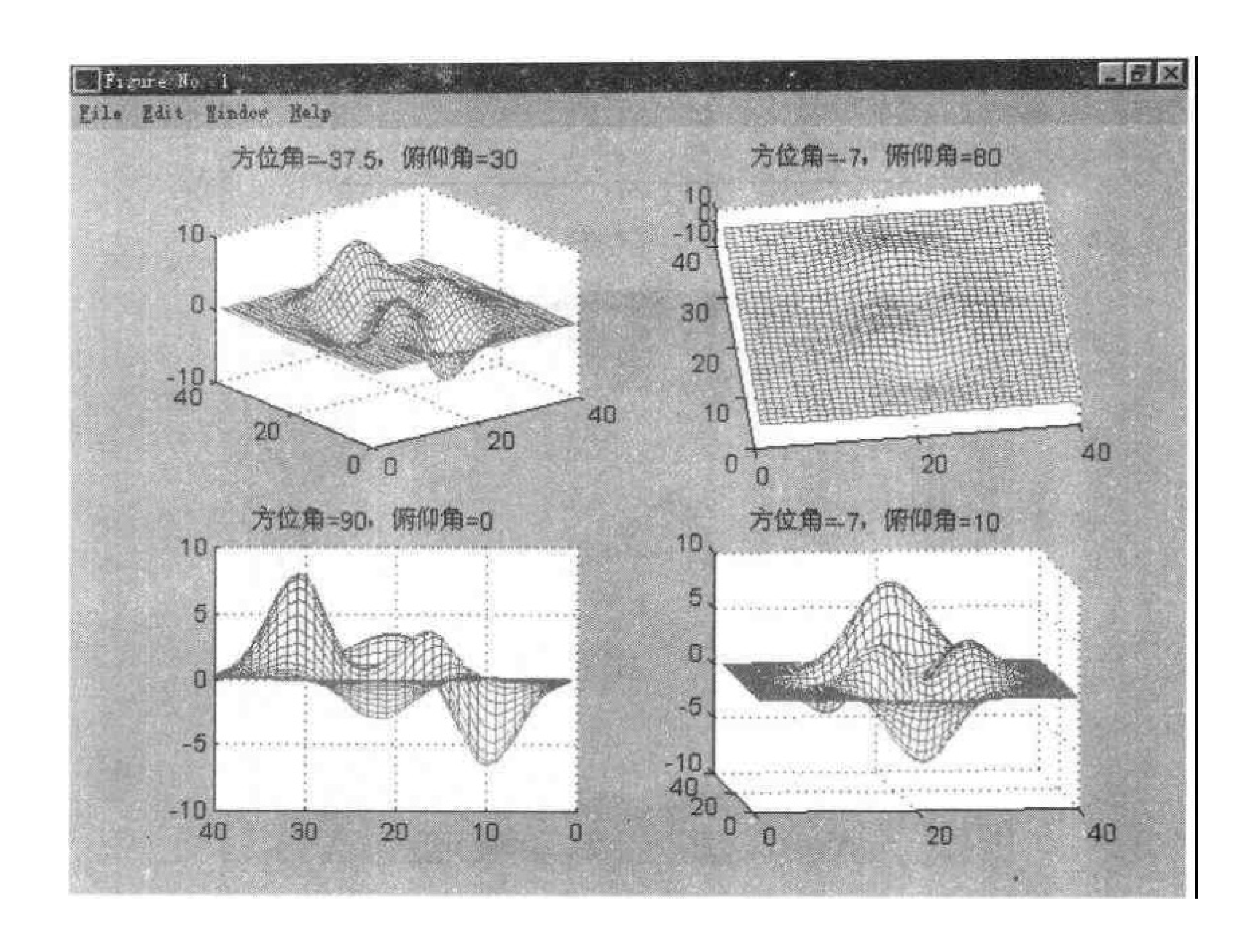


图 4.42　范例 4.21 的运行结果

4.3　图 形 句 柄

MATLAB 提供了一系列用于创建和操作线、面、文字、图像等基本图形对象的底层图形指令．这组指令可用于对图形的各个基本对象进行更为细致的修饰和控制，不仅可以产生更为复杂的图形，而且为动态图形的制作奠定了基础．MATLAB 的这个系统称为句柄图形（handle graphics）．

4.3.1　图形对象

图形对象是 MATLAB 的句柄图形系统中最基本的图元．在句柄图形系统中，所有的图形操作都是针对图形对象而言，底层指令使用户可以对图形的一个或几个对象进行独立的操作，而不影响图形的其他部分．图形对象体系具有树状层次结构，如图 4.43 所示．它反映了图形对象间的相互依赖性，例如，线需要轴作为参照，而轴只存在于图形窗口中．

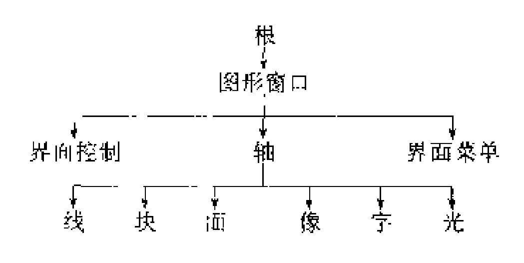


图4.13 图形对象树

根屏幕(root)位于图形对象树的顶层,对应于计算机屏幕,在MATLAB图形系统中只有一个根,其他所有图形对象都是它的后代. 当启动MATLAB时,根屏幕对象就已经存在,你不用去创建它,也不能去破坏它,可以通过set函数和get函数获取根屏幕对象的品性.

图形窗口(figure)是根屏幕上的窗口,窗口的数量不限. 所有的图形窗口都是根屏幕的子代,除根外的其他对象则是窗口的后代. 所有对象创建函数和高层作图指令都会建立一个图形窗口,也可以用figure函数直接创建一个图形窗口,如果没有详细描述figure函数的自变量,则MATLAB用缺省值创建图形窗口. 必须注意,figure(h)函数的执行,必须依赖h值的选取. 如果h是一个已存在的图形的句柄,则figure(h)指令使得该图形成为当前的可视图形,且位于所有图形窗口之上;如果h不是一个已存在的图形的句柄,而是一个整数,则figure(h)指令创建一个新图形,并将h作为其句柄;如果h既不是一个图形的句柄又不是一个整数,则会产生错误.

界面控制(uicontrol)是借助鼠标去执行操作,激活功能的用户界面控制,实现图形化的用户接口. 它是图形窗口的子辈,独立于轴. MTALAB支持九种类型的界面控制,每种都适用于一个不同的用途,例如弹出菜单、按钮、明细表等.

界面菜单(uimenu)是建立在图形窗口上方的一系列用户界面菜单和子菜单. 它也是图形窗口的子辈,独立于轴.

线(line)是创建大多数二维图形和一部分三维图形的基本图元. Line函数在当前轴上创建线对象,并且可详细说明线型、颜色、宽度等特征.

块(patch)是填充多边形,用单色或插补色进行渲染,可详细说明块对象的色彩和亮度. 它是轴的子辈,其位置决定于轴所建立的坐标系.

面(surface)是矩阵数据的三维空间表现,由许多四边形组成,四边形的顶点又由所给的数据定位. 面可以用单色或插补色表现,也可以用点间连线表示. 它是轴的子辈,其位置决定于轴所建立的坐标系.

像(image)是矩阵元素直接映射到当前的色图上所得的结果. 像是一个二维图形,没有观察角的调整问题,有自己的色图. 它是轴的子辈,其位置决定于轴所建立的坐标系.

字(text)是字符串. 它是轴的子辈,其位置决定于轴所建立的坐标系.

光(light)说明了影响所有在轴内的对象的光源. 光对象本质上看不见,但可以通过设置块对象和面对象的品性来观察它在块对象和面对象上的影响,它是轴的子辈,其位置决定于轴所建立的坐标系.

4.3.2 图形对象的句柄

每个具体的图形对象在它创建时就被赋以一个惟一的识别标记,这就是该图形对象的句柄. 有一些曲线图,例如多条等位线中的每一条都有自己的句柄.

根屏幕的句柄总是零，图形窗口的句柄为一整数，其缺省值显示在窗口标题上．而其他对象的句柄为浮点数．通常，图形对象的句柄被赋予变量，以备使用，而不是直接从键盘输入从屏幕所见句柄．

MATLAB 提供了三个专门用于获取对象句柄的函数：

Gcf　　返回当前图形窗口的句柄．

Gca　　返回当前轴的句柄．

Gco　　返回当前对象的句柄．

它们既可以直接赋值给某一个变量，也可以作为其他函数的输入变量使用．例如指令 delete (gca) 将删除当前轴以及它的所有子代．

4.3.3 对象创建函数

在 MATLAB 中，除根屏幕 Root 外，所有的对象都由与之同名的内部函数 (Build-in Functions) 创建．这些内部函数是 MATLAB 的作图核心程序，是建立 MATLAB 的高层作图函数的基础．例如 plot 和 surf 都是调用相应的底层函数来绘制图形的．以下内容简要列出了底层函数的功能及调用方式，每一个函数都返回相应的句柄．

(1) 创建图形对象 (figure)

调用格式有四种，如图 4.44 所示．

```
figure
figure('PropertyName',PropertyValue,...)
figure(h)
h=figure(n)      n 为整数
```

图 4.44　调用格式

例 4.22　输入如图 4.45 所示的程序．

```
scrsz=get(0,'ScreenSize');
figure('Position',[1 scrsz(4)/2 scrsz(3)/2 scrsz(4)/2])
```

图 4.45　范例 4.22

就可以在屏幕的左上角看见一个图形窗口，是当前屏幕尺寸的四分之一大小．在程序中，运用了根屏幕对象的 "ScreenSize" 品性去定义图形窗口大小．

(2) 创建轴对象 (axes)

调用格式有四种，如图 4.46 所示．

```
axes
axes('PropertyName',PropertyValue,...)
axes(h)
h=axes(...)
```

图 4.46　调用格式

例 4.23　输入如图 4.47 所示的程序．

```
axes('position',[.1 .1 .8 .6])
mesh(peaks(20));
axes('position',[.1 .7 .8 .2])
pcolor([1:10;1:10]);
```

图 4.47　范例 4.23

就可以在屏幕上看见一个包含两个不同轴对象的图形，第一个图形占据了距图形窗口底部三分之二的面积，第二个图形占据了距图形窗口顶部三分之一的面积，如图 4.48 所示.

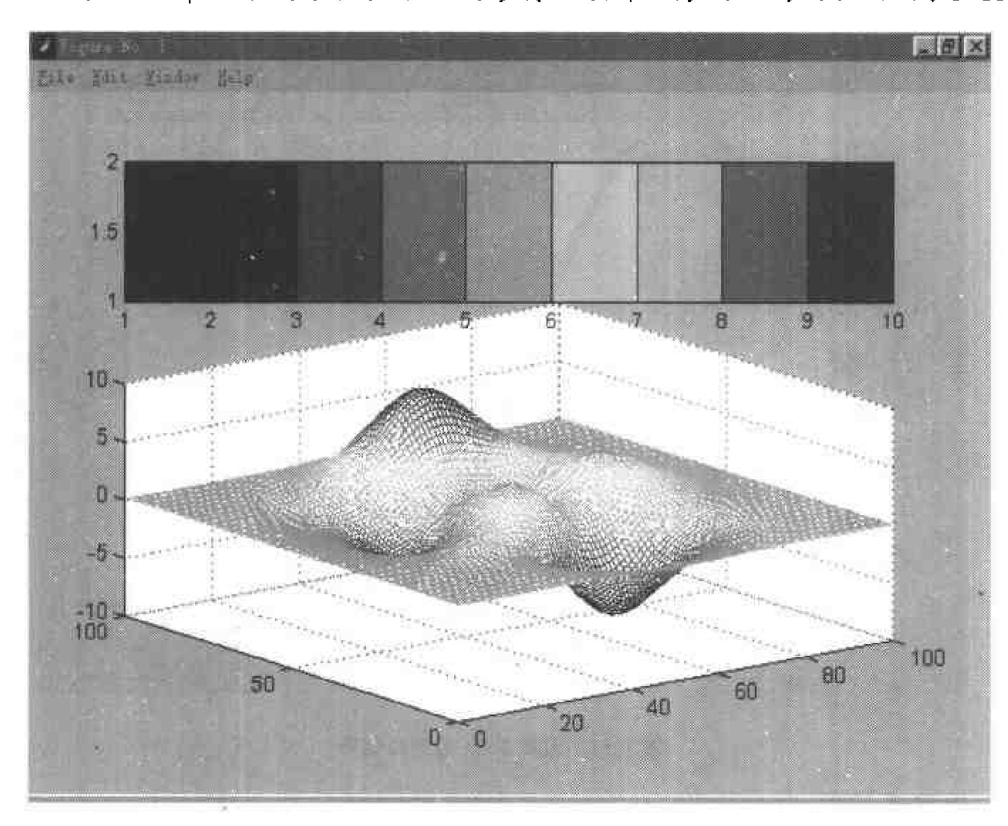


图 4.48　范例 4.23 的运行结果

(3) 画线 (line)

调用格式有五种，如图 4.49 所示.

```
line(X,Y)
line(X,Y,Z)
line(X,Y,Z,'PropertyName',PropertyValue,...)
line('PropertyName',PropertyValue,...)
h=line(...)
```

图 4.49　调用格式

例 4.24　输入如图 4.50 所示的程序.

```
t=0:pi/20:2*pi;
hline1=plot(t,sin(t),'k');
hline2=line(t+.06,sin(t),'LineWidth',4,'Color',[.8.8.8]);
set(gca,'Children',[hline1 hline2])
```

图 4.50　范例 4.24

就可以在屏幕上看见一条正弦曲线及一条亮灰色的正弦曲线阴影，并且正弦曲线位于阴影曲线前方，如图 4.51 所示.

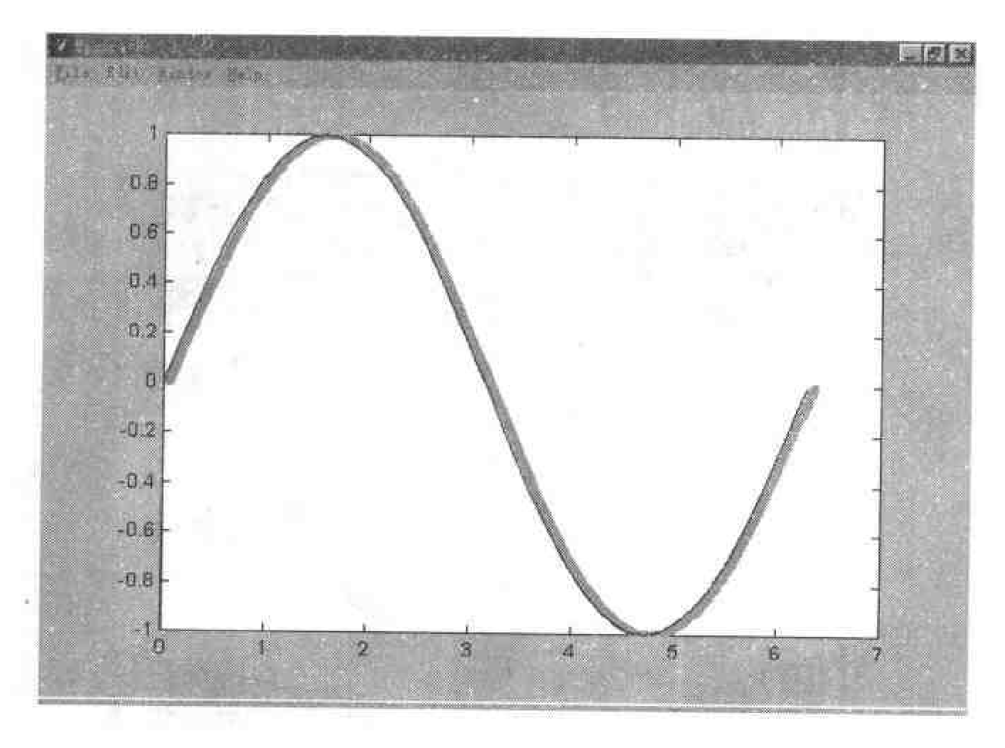


图 4.51　范例 4.24 的运行结果

(4) 填充多边形 (patch)

调用格式有五种，如图 4.52 所示.

```
patch(X,Y,C)
patch(X,Y,Z,C)
patch(...'PropertyName',PropertyValue...)
patch('PropertyName',PropertyValue...)
handle=patch(...)
```

图 4.52　调用格式

例 4.25　输入如图 4.53 所示的程序.

```
x=[0 1;1 1;0 0];
y=[2 2;2 1;1 1];
z=[1 1;1 1;1 1];
tcolor(1,1,1:3)=[1 1 1];
tcolor(1,2,1:3)=[0.7 0.7 0.7];
patch(x,y,z,tcolor)
```

图 4.53　范例 4.25

就可以在屏幕上看见两个三角形，各有三个顶点，上部三角形填充白色，下部三角形填充灰色，如图 4.54 所示．

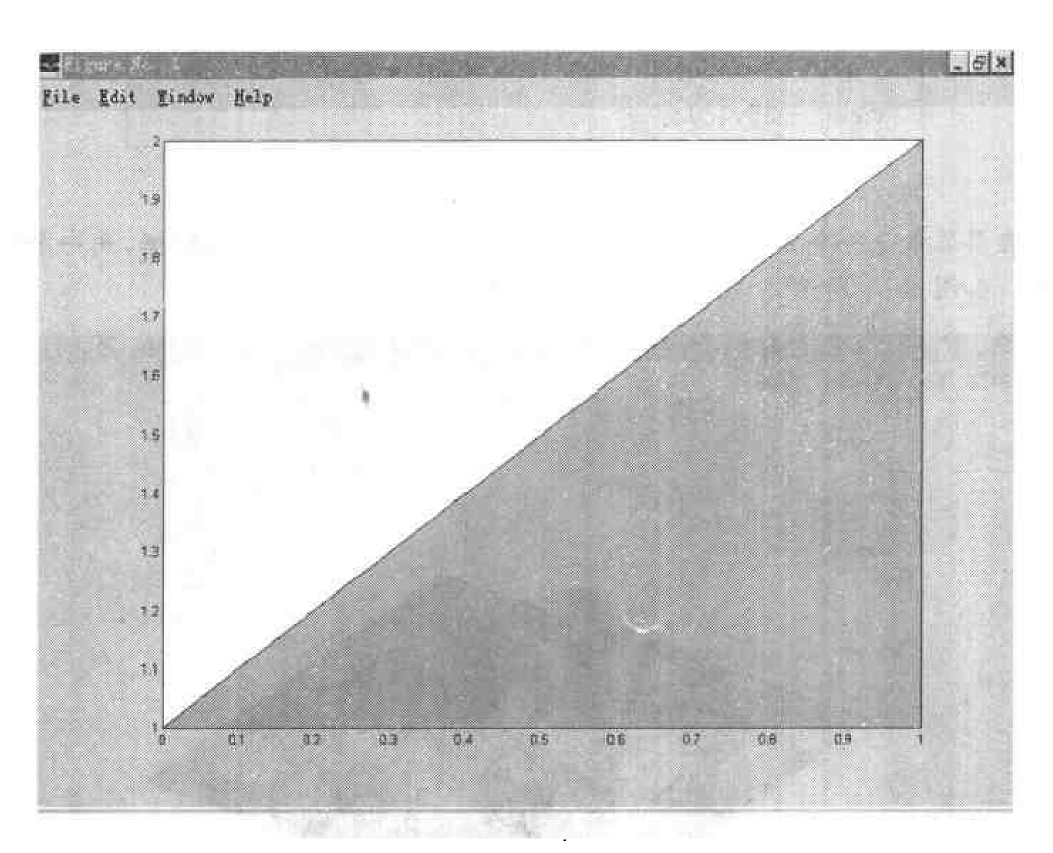


图 4.54　范例 4.25 的运行结果

(5) 创建三维曲面 (surface)

调用格式有五种，如图 4.55 所示．

```
surface(Z)
surface(Z,C)
surface(X,Y,Z)
surface(X,Y,Z,C)
surface(...'PropertyName',PropertyValue,...)
h=surface(...)
```

图 4.55　调用格式

例 4.26　输入如图 4.56 所示的程序.

```
load clown
surface(peaks,flipud(X),...
'FaceColor','texturemap',...
'EdgeColor','none',...
'CDataMapping','direct')colormap(map)
view(-35,45)
```

图 4.56　范例 4.26

就可以在屏幕看见一个三维曲面，它用 MATLAB 的 peaks 文件产生数据，用一个小丑肖像填色，如图 4.57 所示.

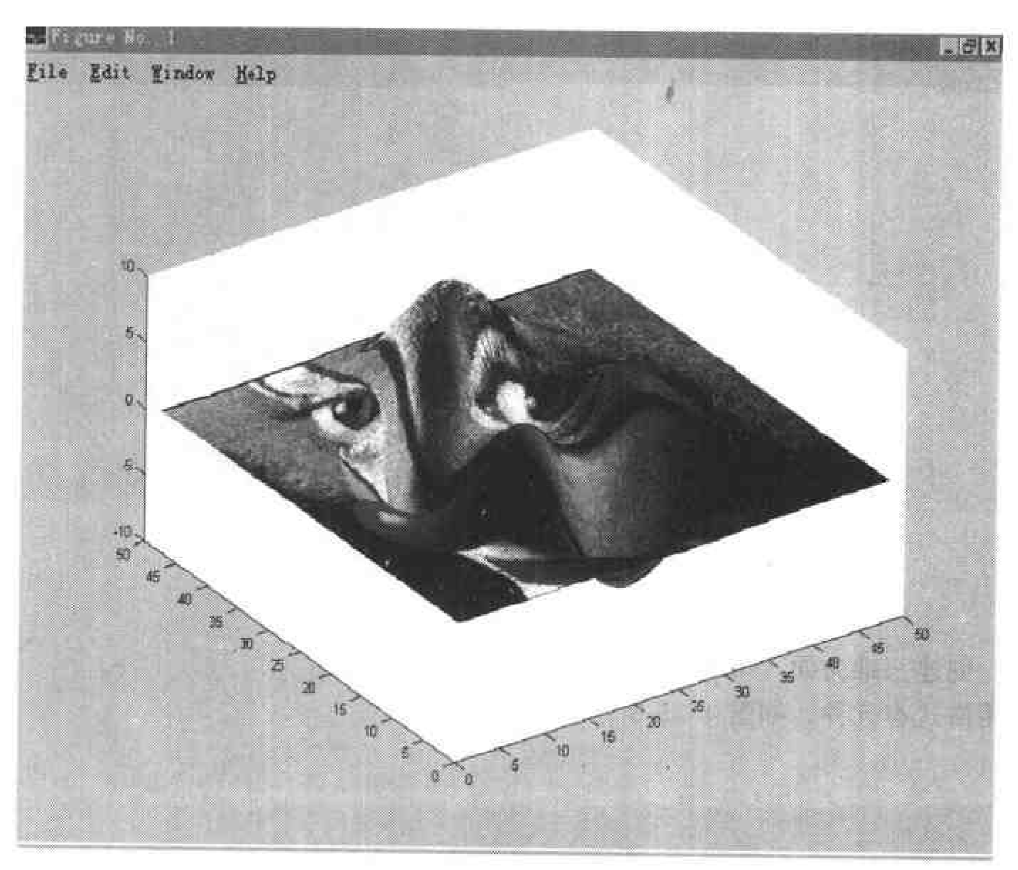


图 4.57　范例 4.26 的运行结果

(6) 显示图像 (image)

调用格式有五种，如图 4.58 所示.

```
image(C)
image(x,y,C)
image(...,'PropertyName',PropertyValue,...)
image('PropertyName',PropertyValue,...)
handle=image(...)
```

图 4.58 调用格式

例 4.27 输入如图 4.59 所示的程序.

```
load durer
whos
image(X)
colormap(map)
axis image
```

图 4.59 范例 4.27

就可以在屏幕上看见一幅图像. 其中，图像矩阵的大小由载入的文件决定，而矩阵元素又决定了图像的颜色与明暗，如图 4.60 所示.

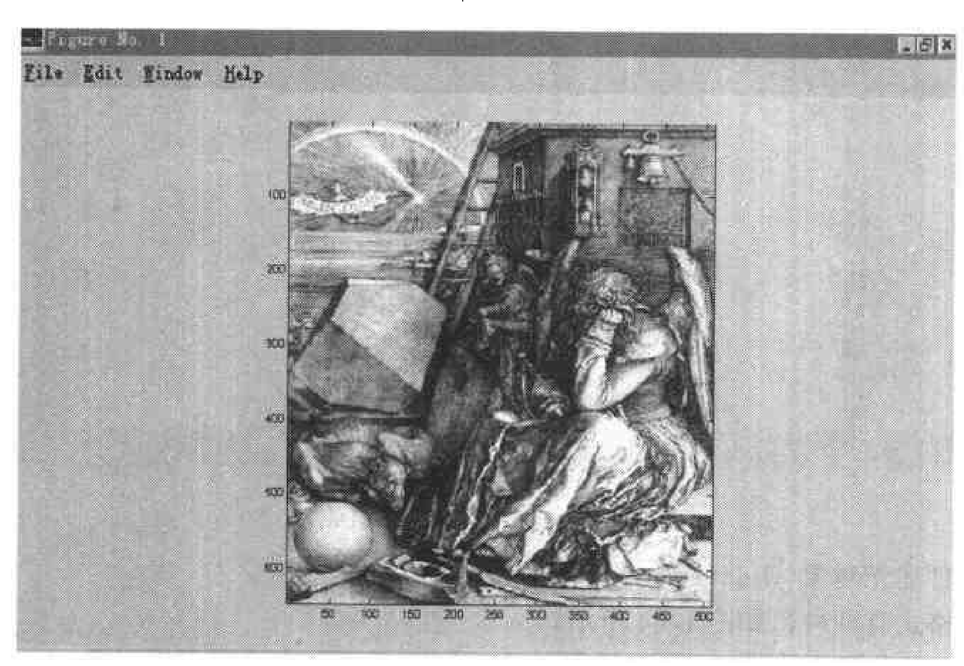


图 4.60 范例 4.27 的运行结果

标注文字 (text)

调用格式有四种，如图 4.61 所示.

```
text(x,y,'string')
text(x,y,z,'string')
text(...'PropertyName',PropertyValue...)
h=text(...)
```

图 4.61　调用格式

例 4.28　输入如图 4.62 所示的程序.

```
plot(0:pi/20:2*pi,sin(0:pi/20:2*pi))
text(pi,0,' \leftarrow sin(\pi)','FontSize',30)];
```

图 4.62　范例 4.28

就可以在屏幕上看见一条正弦曲线，其标注文字为 sin(π)，并由左箭头指向曲线，如图 4.63 所示.（这种用法在本章前面已介绍过，这里进一步介绍，是从图形句柄应用的完整性考虑的.）

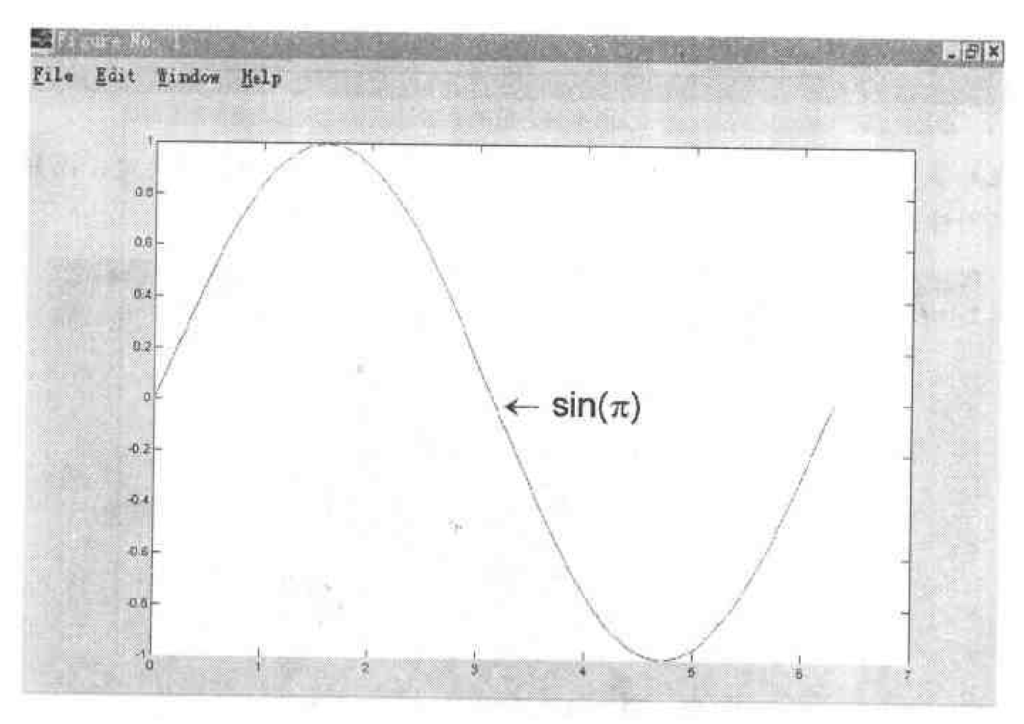


图 4.63　范例 4.28 的运行结果

(7) 创建光对象（light）

调用格式有两种，如图 4.64 所示.

```
light('PropertyName',PropertyValue,...)
handle=light(...)
```

图 4.64　调用格式

例 4.29　输入如图 4.65 所示的程序.

```
h=surf(peaks);
set(h,'FaceLighting','phong','FaceColor','interp',...
  'AmbientStrength',0.5)
light('Position',[1 0 0],'Style','infinite');
```

图 4.65 范例 4.26

就可以在屏幕上看见一个光源照亮曲面，所定义的光源位于无穷远，其发光方向由向量 [1 0 0] 设定，即沿着 x 轴方向发射，如图 4.66 所示.

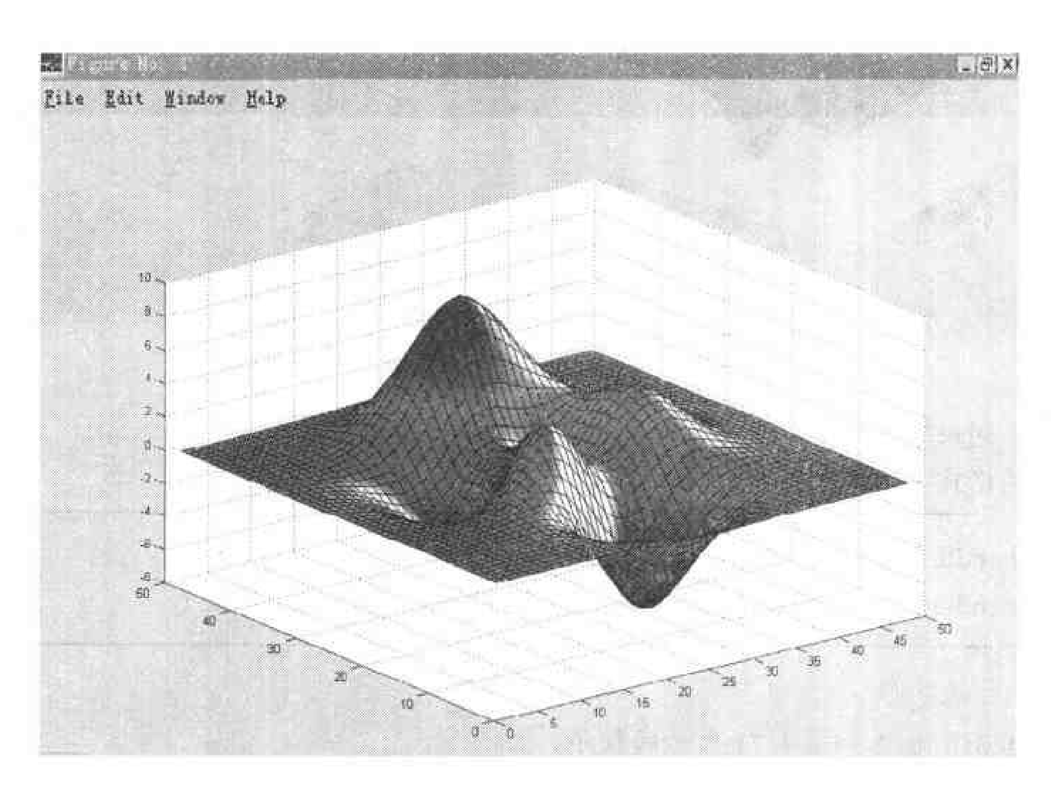


图 4.66 范例 4.29 的运行结果

(8) 用户界面控制 (uicontrol)

调用格式有四种，如图 4.67 所示.

```
handle=uicontrol(parent)
handle=uicontrol(...,'PropertyName',PropertyValue,...)
```

图 4.67 调用格式

例 4.30 输入如图 4.68 所示的程序.

```
h=uicontrol('Style','Pushbutton','Position',...
[1 1 50 50], 'Callback','cla','String','Clear');
```

图 4.68 范例 4.30

就可以在屏幕的左下角看见一个“clear”按钮，选择该按钮，则清除当前轴上的图形对象，如图 4.69 所示．

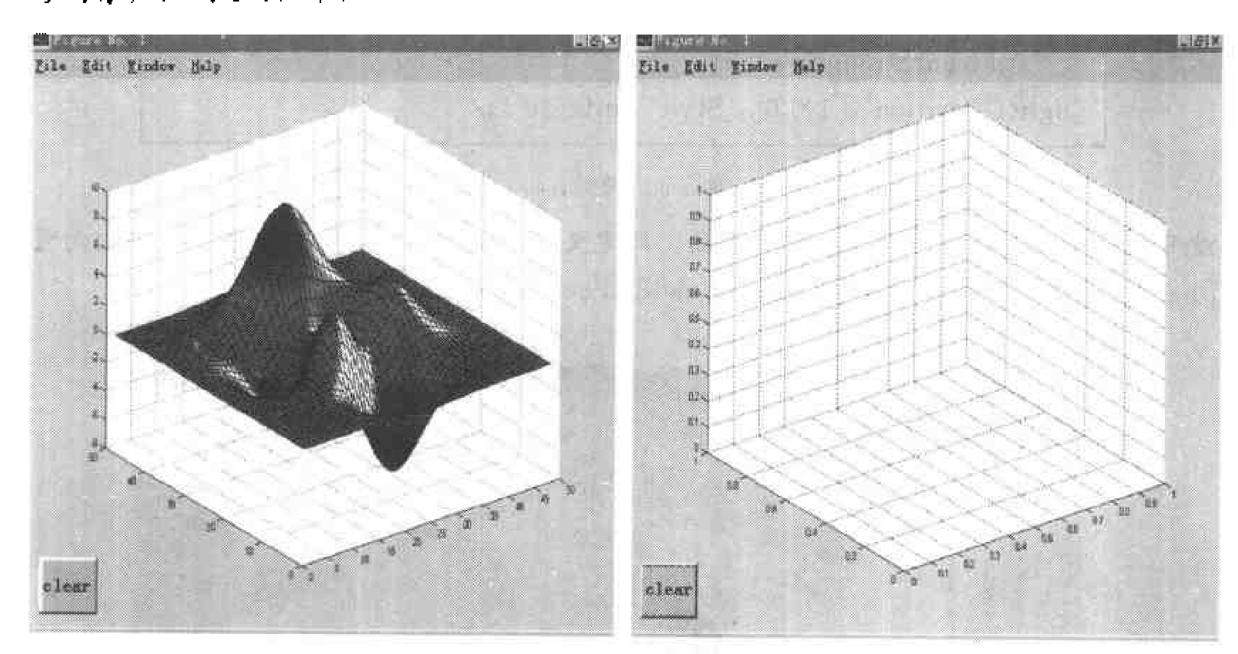


图 4.69 范例 4.30 的运行结果

(9) 用户界面菜单（uimenu）

调用格式有四种，如图 4.70 所示．

```
handle=uimenu('PropertyName',PropertyValue,...)
handle=uimenu(parent,'PropertyName',PropertyValue,...)
```

图 4.70 调用格式

例 4.31 输入如图 4.71 所示的程序．

```
f=uimenu('Label','Workspace');
    uimenu(f,'Label','New Figure','Callback','figure');
    uimenu(f,'Label','Save','Callback','save');
    uimenu(f,'Label','Quit','Callback','exit',...
        'Separator','on', 'Accelerator', 'Q');
```

图 4.71 范例 4.31

就可以看见在图形窗口的菜单栏里增加了一个标注为“workspace”的菜单，其下拉菜单有三个，选择“new figure”，可以创建一个新的图形窗口；选择“save”，存贮 workspace 变量；选择“exit”或者快捷键“ctrl+q”，则系统退出 MATLAB，如图 4.72 所示．

每个内部函数只能创建一个图形对象，并将其置于适当的父辈对象中．例如，line 函数的执行是在当前轴上利用缺省特性数据画线．如果此指令运行前，“轴”、“窗”不存在，则 MATLAB 会自动创建它们；如果此指令运行前，“轴”、“窗”已存在，则线将被画在

已有的轴上，且不影响该轴上已有的其他对象．

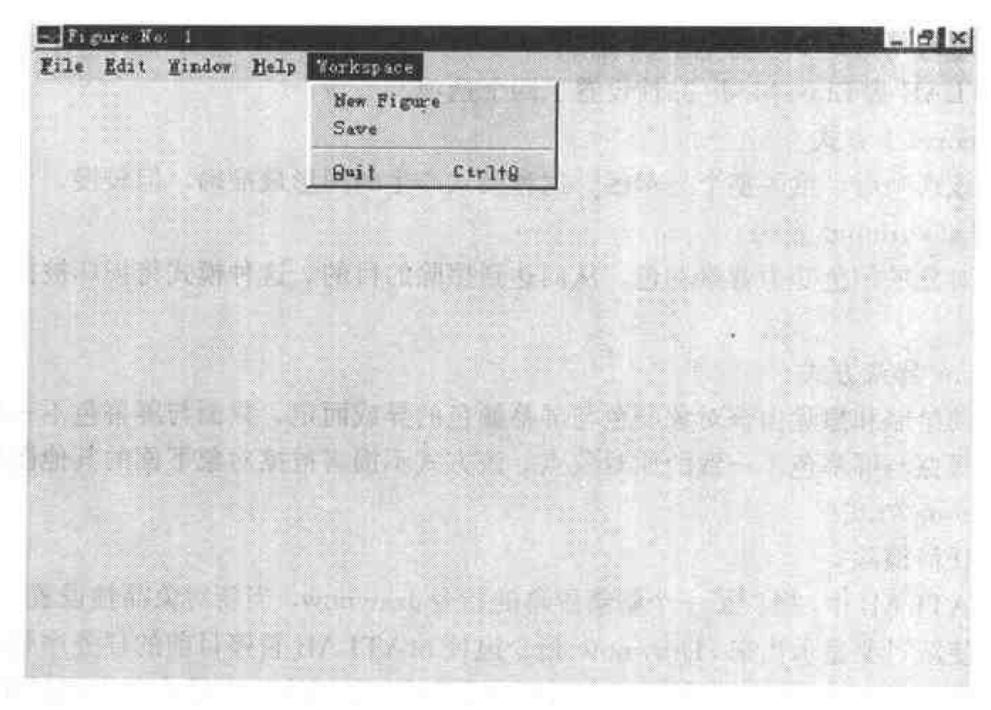


图 4.72 范例 4.31 的运行结果

4.3.4 对象品性及其设置和查询

所有的对象都具有控制其如何表现的品性．对象品性可分为两类：一类是“共性”，包括类型（Type）、是否可视（Visible）、剪接（Clipping）、中断允许（Interruptible）等；另一类是“特性”，如“轴”的刻度、定义“面”的数据等．

当对象被创建时，该对象的品性就被一组缺省值初始化．

MATLAB 提供了两种方式来设置品性值，可以在对象创建时设置其品性，或者通过函数 set 重新设置已存在的对象的品性．例如：

set（h） 显示 h 对象所有可设置的品性及取值．

set（h,'PropertyName'） 显示 h 对象（PropertyName）指定品性的可取值．

set（h,'PropertyName', PropertyValue, ...） 设置 h 对象（PropertyName）指定品性的值．

MATLAB 也提供了 get 指令查询任何品性值．例如：

get（h） 查询 h 对象所有品性的当前值．

get(h,'PropertyName') 查询 h 对象(由 PropertyName)所指定品性的当前值.

4.3.5 实时动画的制作

图形从静到动是质的飞跃．实时动画的制作要求用户具有全面的知识和灵感，同时对软硬件资源也提出了较高的要求．一般来说，实时动画的制作需要保持整个背景图案不变，只更新运动部分的图案，以便加快整幅图像的实时生成速度．

在实时动画制作中，改变某几个图形对象面不破坏其他部分图形的具体方法是：

1）计算活动对象的新位置，并在新位置上将它显示出来；2）擦除原位置上原有对象，刷新屏幕；3）重复步骤1和步骤2. 在MATLAB图形系统中，只需在创建图形对象时指定它的擦除方式（EraseMode）便可以擦除旧对象，显示新对象，并且不破坏背景图案.

MATLAB为EraseMode品性设置了四个选项：

（1）normal 方式

使用该选项后，重画整个显示区. 这种模式产生的图形最准确，但较慢.

（2）background 方式

将旧对象的颜色变为背景颜色，从而达到擦除的目的. 这种模式将损坏被擦对象下面的对象.

（3）xor 异或方式

对象的绘制和擦除由该对象颜色与屏幕颜色的异或而定. 只画与屏幕色不一致的新对象点；只擦与屏幕色不一致的原对象点. 该方式不损害被擦对象下面的其他图像.

（4）none 方式

不做任何擦除.

在MATLAB中，提供了一个刷新屏幕的指令drawnow. 当新对象品性设置后，应刷新屏幕，使新对象显示出来. Drawnow指令迫使MATLAB暂停目前的任务序列而去刷新屏幕；若没有Drawnow指令，MATLAB要等任务序列执行完后才会去刷新屏幕.

例 4.32　输入如下的程序.

```
t=0: pi/48: 4*pi;
y=sin (t);
plot (t, y,'b')
n=length (t);
h=line ('color', [0 0.5 0.5],'linestyle','.', ...
  'markersize', 50,'erasemode','xor');
i=1;
while 1
     set (h,'xdata', t (i),'ydata', y (i));
     drawnow;
     i=i+1;
     if i>n,
          i=1;
     end
end
```

就可以在图4.73中看见一个小球沿正弦曲线的轨迹移动. 小球移动的速度决定于 t 的增量、计算机运行窗口的多少以及计算机的运算速度. 由于程序设为无穷循环，可用Ctrl+Break中断小球的移动.

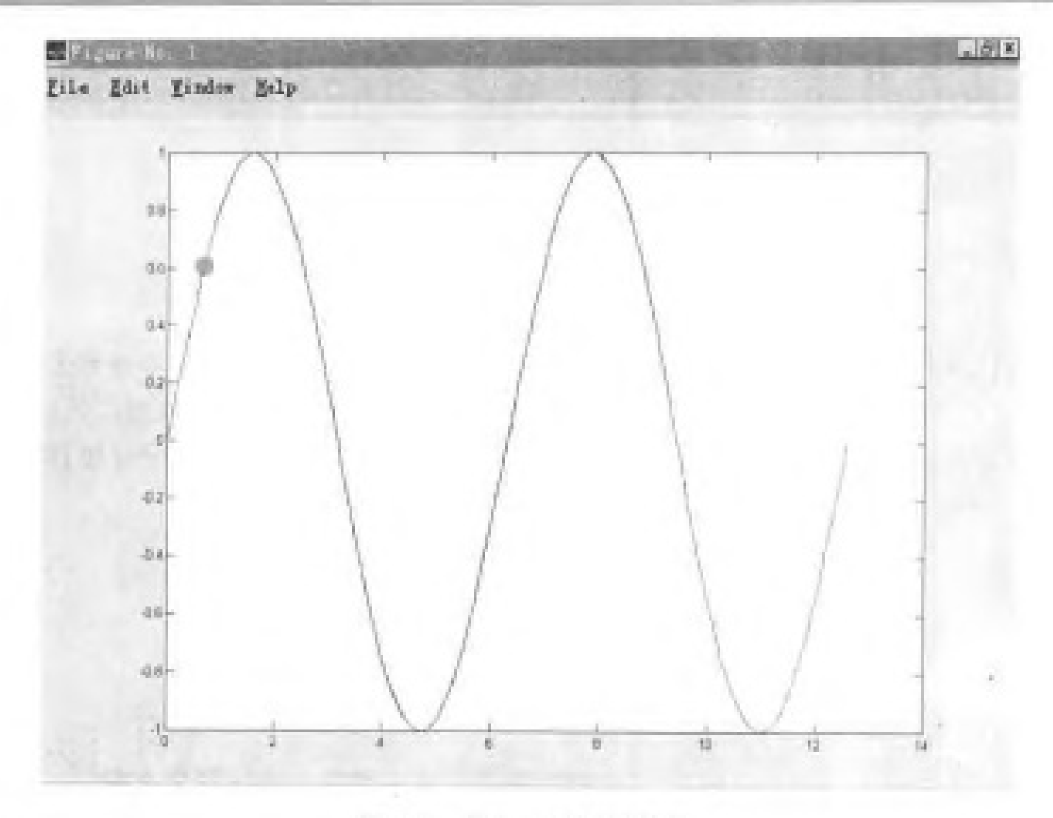


图 4.73　范例 4.32 的运行结果

第五章　MATLAB 的程序设计

MATLAB 语言编写的简单程序在前面几章中已经接触过，本章将系统地叙述 m 文件功能、形式、程序结构、数据结构、变量定义和程序流控制，函数调用，数据的输入输出；本章还给出一些 MATLAB 语言设计的神经网络模型和训练的例子（如 BP 网、模糊神经网络等）.

5.1　MATLAB 程序设计入门

5.1.1　编辑程序和 m 文件的形式

和其他程序语言一样，MATLAB 程序也可以用一般的文字编程器来编写，如 MS-WORD，PE 等都可以，但必须要将编写的程序保存成文本文件. 此外，在 MATLAB 的命令行上也可以输入 EDIT 命令来编辑文件，例如：

edit 或 edit dddd

编辑之后，文件名会被自动保存为 dddd. m. 也可以用鼠标在命令窗口上点击 File 菜单中选择 New 中的 m 文件，如图 5.1 所示.

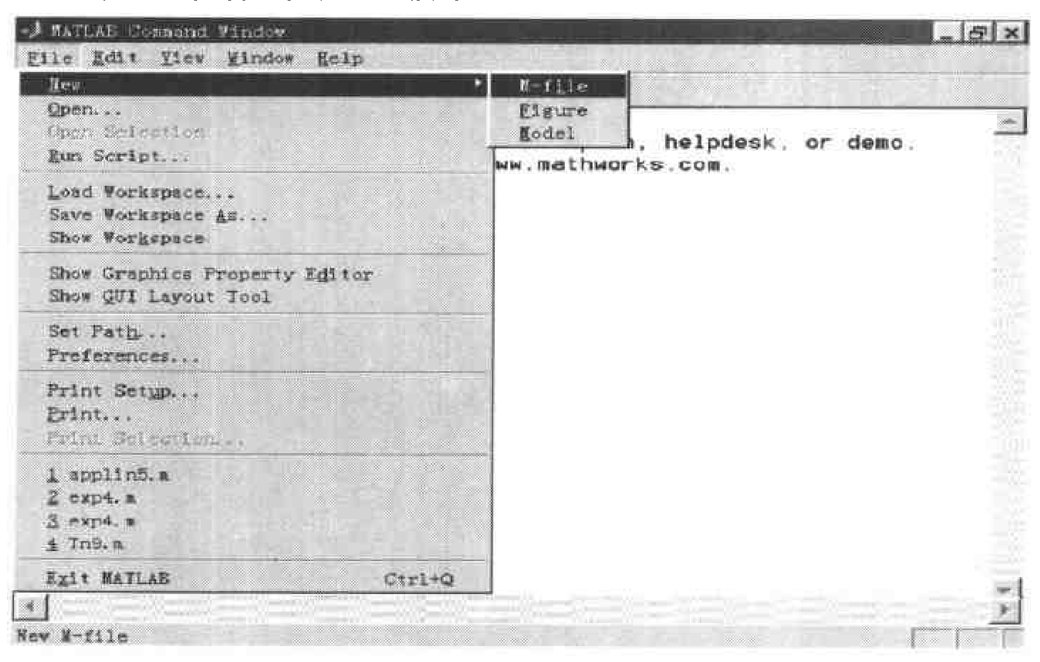


图 5.1　启动 MATLAB 编辑器建立 m 文件

m 文件有两个形式：一种称为命令文件 (script file)，就好像 dos 下的批处理文件一样. 这类程序包含了一连串的 MATLAB 命令，执行时依序执行. 当然，当用户要运行的

MATLAB 指令较多时，直接从键盘上逐行输入指令不是不可以，但显得比较麻烦；而命令文件则可以较好地解决这一问题，用户可以将一组相关命令编辑在同一个命令文件中，运行时只要输入文件名，MATLAB 就会自动按顺序执行文件中的命令. 另一种称为函数文件（function file），它的第一句可执行语句是以 function 引导的定义语句，在函数文件中的变量都是局部变量. 两者间的比较见表 5.1.

表 5.1　两种 m 文件的比较

m 文件	命　令　文　件	函　数　文　件
参数	没有输入参数，也不会返回参数	可以接受参数，也可以返回参数
数据	处理的数据即为命令区的数据	函数里面的变量为局部变量，但也可以设外部变量
应用	经常用在一连串费时的指令上	扩充 MATLAB 函数库，以及特殊的应用上

5.1.2　MATLAB 的命令文件

命令文件实际上只包括两部分，即注解和指令. 注解部分开头必须用百分比符号％注明. 命令文件中的语句以访问 MATLAB 工作间（workspace）中的所有数据. 运行过程中的所有变量均是全局变量，这些变量一旦生成，就一直保存在内存空间中，除非用户运用 clear 指令将它们清除.

运行一个命令文件等价于从指令窗（command window）中按顺序连续运行文件里的指令. 由于命令文件只是一串指令的集合，因此程序不需要预先定义，而只需按在指令窗口中的指令输入顺序依次将指令编辑在命令文件中就可以了.

例 5.1　画出花瓣形状的图案.

1）运用 MATLAB 编辑器，建立 m 文件，如图 5.2 所示.

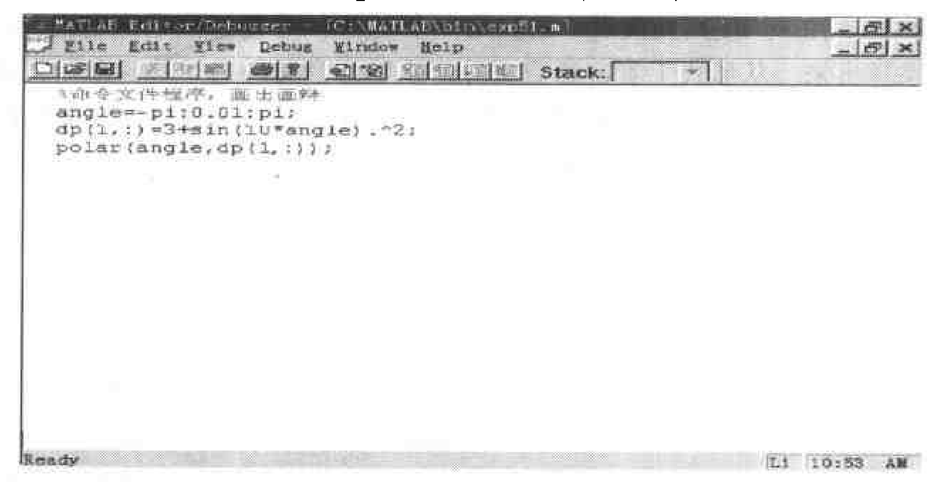


图 5.2　范例 5.1

2）选择[file]下拉菜单中[save]子项，将所写文件存放于磁盘中，并起名为 exp51.m.

3）在 MATLAB 的指令窗中键入文件名 exp51，运行该文件的结果就可以在屏幕上显示出来，如图 5.3 所示.

本例 m 文件说明：

1）符号“％”引的行是注释行，不予执行，可有可无.

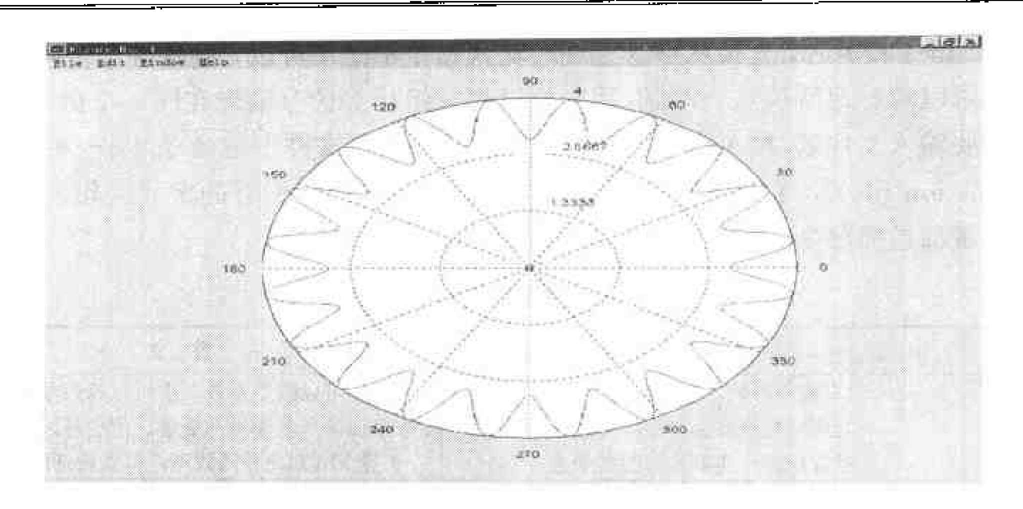


图 5.3 范例 5.1 的运行结果

2）不需要"end"作为命令文件的结束标志.

3）exp51.m 运行后存放在内存中的变量可以用 who 指令看到.

4）如果程序保存在其他目录，可以利用 File 菜单中的 Set Path 窗口先将工作目录设为程序所在的目录，然后在命令行中输入 exp51 即可执行. 最简单方法：如果 exp51.m 存放在自己的工作目录（假如名为 d：\mywork）上，那么在运行 exp51.m 之前，应该先在 MATLAB 指令窗中先运行 cd d：\mywork.

5.1.3 MATLAB 的函数文件

如果 m 文件的第一行包含 function，此文件就是函数文件. 每一个函数文件都定义一个函数. 事实上，MATLAB 提供的函数指令大部分都是由函数文件定义的. 这足以说明函数文件的重要性. 从使用角度看，函数是一个"黑箱"，把一些数据送进去，经加工处理，把结果送出来. 从形式上看，函数文件区别于命令文件之处是：命令文件的变量在文件执行完后保留在内存中；而函数文件内定义的变量仅在函数文件内部起作用，当函数文件执行完后，这些内部变量将被消除.

MATLAB 函数文件实际上包含五个部分：

1）函数定义行；

2）函数主体；

3）H1 行；

4）函数说明；

5）注解.

其中，完整的函数一定要包括函数定义行与函数主体，其余三个部分都是为了辅助使用的，图 5.4 即为一个函数的例子，它接受一个参数，并且执行 comp(x)=(x+10)^2*10 运算，然后返回一个值. 在函数定义行上，除了 function 为关键字外，返回值就直接定义成 y=comp（x）.

如果要执行这个函数，可以在命令行上直接输入：

```
exp52 (10)
```

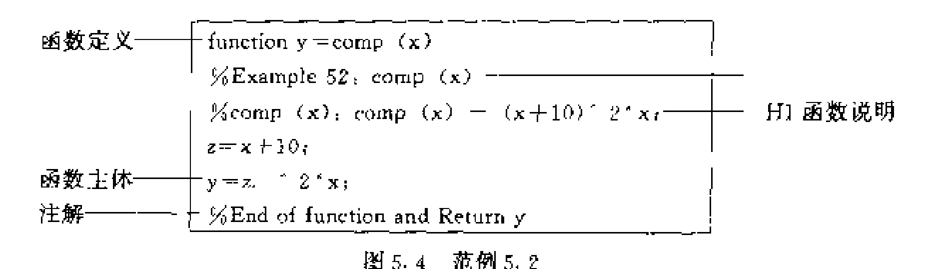


图 5.4　范例 5.2

会得到结果：

```
ans =
        4000
```

或输入：

```
k=10;
exp52 (k)
```

得到结果：

```
ans =
        4000
```

说明：

函数文件就像 MATLAB 提供的函数库一样，读者也可以建立自已的函数库及其说明文件，也可以为每一个函数编写一个使用说明. 只要遵循 MATLAB 的规定，就可以很简单地实现这种功能，这个规定如下：

1）将自己的函数程序保存在同一个目录下.

2）第一行执行指令（定义行）的作用：指明该文件是函数文件、定义函数名、输入参数和输出参数.

3）每个函数库的 H1 必须紧跟在定义行之后.

4）每个函数库的使用说明紧跟在 H1 之后.

5）在 m 文件前面，连续几行带符号“%”的注释行有两个作用：一是随 m 文件全部显示打印时，直接起解释提示作用；二是供 help 指令在线查询用.

5.2　参数与变量

5.2.1　参数

功能强大的程序往往需要接受不同数目的参数输入，功能更强大的程序还要接受不同类型的参数输入. MATLAB 在这方面提供了非常方便的界面. 如下述程序（例 5.2），这个文件原本设定有三个输入参数 (a, b, c) 以及一个输出参数. 但是通过 nargin (MATLAB 本身自带的函数指令）内部函数检查实际输入参数的数目，在程序内就可以决定不同的运算方式和输出，其执行结果如图 5.5 所示.

例 5.2　程序：

```
function out=check_arg (a, b, c)
%Example 5-3: check_arg
```

```
%check _arg (a, b, c), the return value is decided by the function
%input number and input arguments.
if (nargin==1)
    out=a;
elseif (nargin==2)
   out= (a.^2+b.^2);
elseif (nargin==3)
    out= (a*b*c) /2;
end
```

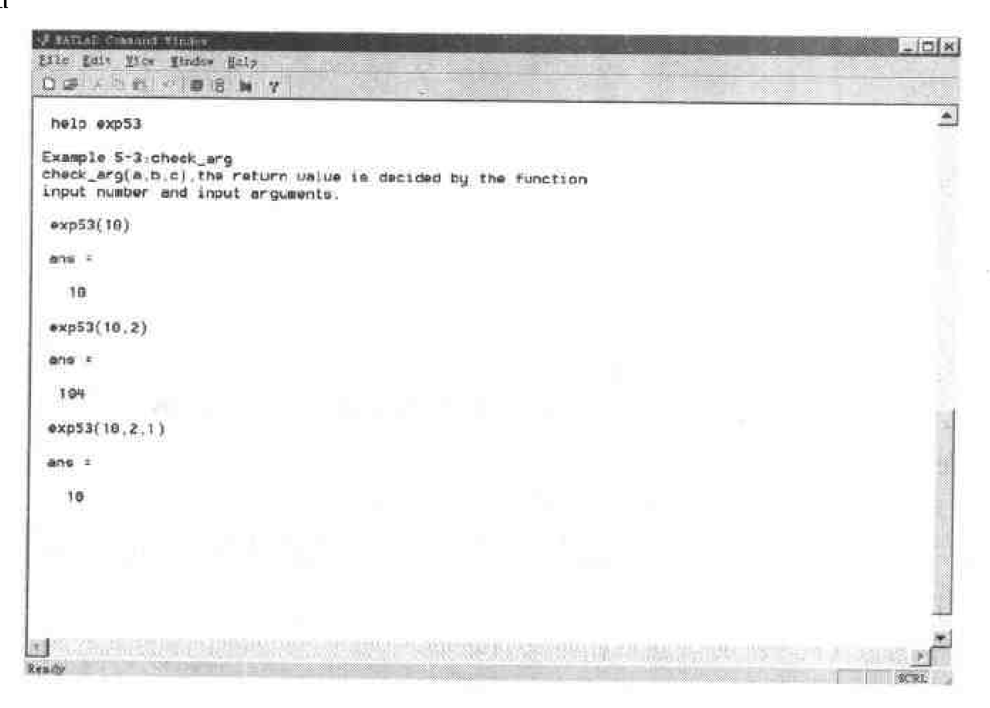


图 5.5　范例 5.2 程序的运行结果

当然也可以应用于矩阵参数的输入，如：如下程序（例 5.3）．注意，这个例子中并没有指定输入的数据类型，因此可以接受各种不同的输入．图 5.6 为一个以矩阵输入的结果．

例 5.3　程序

```
function out=check _array _arg (a, b, c)
%Example 5-4: check _array _arg
%check _array _arg (a, b, c), the return value is decided by the function
%input array number and input arguments.
if (nargin==1)
   out=a;
elseif (nargin==2)
   out= (a+b);
elseif (nargin==3)
  out= (a*b*c) /2;
end
```

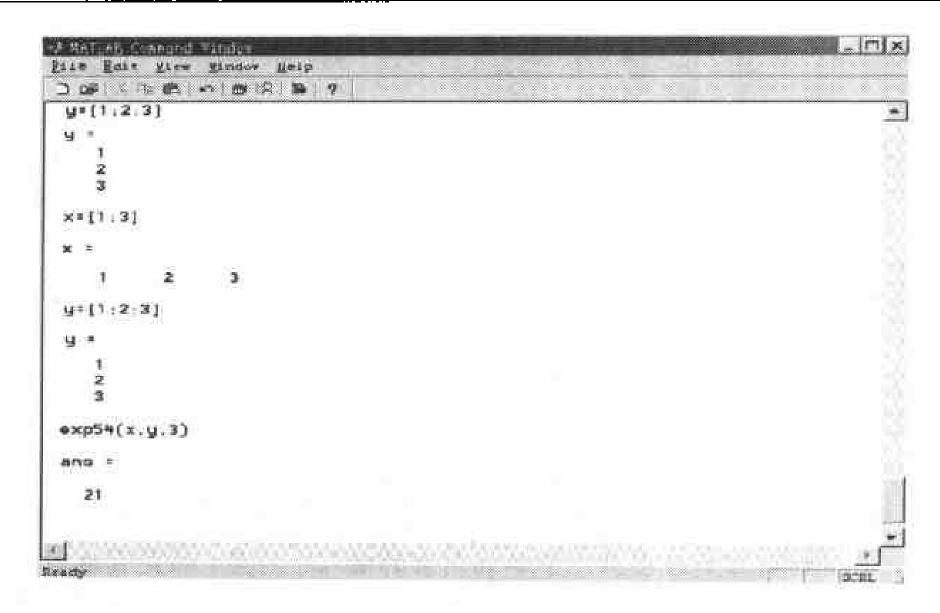


图 5.6 例 5.3 程序的运行结果

注意：如果以数组为输入参数，就需要注意数组运用时的维数.

输出方面也有类似的内部函数，即 nargout. 下面以一个字符串处理函数 token 为例，其程序如下：

例 5.4 程序

```
function [first, remainder] =token (string, separators)
%Example 5-4: token
% [first, remainder] =token (string, separators): Separate the
%first token from input string according to the separators
%char assigned.
if nargin<1, error ('Lack of enough input arguments.'); end
first= []; remainder= [];
len=length (string);
if len==0
   return
end
if (nargin==1)
   separators= [9: 13 32];
  %Default separator is white apace characters
end
i=1;
while (any (string (i) ==separators))
   i=i+1;
   if (i>len), return, end
end
start=i;
```

```
while (~any (string (i) ==separators))
   i=i+1;
if (i>len), break, end
end
finish=i-1;
first=string (start: finish);
if (nargout==2)%if the output number needed is two
   remainder=string (finish+1: length (string));
end
```

例 5.5 程序接受两个参数，一个为字符串的输入，另一个为每个单词间的分隔符. 返回值也有两个参数，第一个返回参数 first 的内容是输入字符串中的第一个单词，第二个返回参数 remainder 的内容是除去第一个单词剩余的输入字符串，而是否返回 remainder，是通过 nargout 是否等于 2 来决定的，在 MATLAB 中，字符串的输入是以引号'' 表示的. 例如：

s='I am a doctor'

就是令 s 等于字符串，图 5.7 为运行结果.

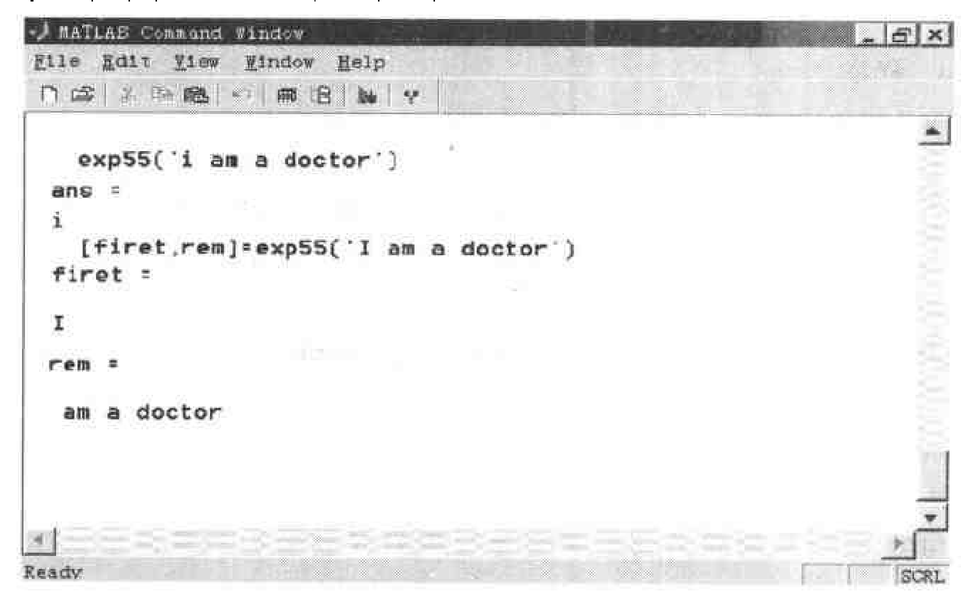


图 5.7 例 5.5 程序的运行结果

5.2.2 局部变量与全局变量

MATLAB 对变量名称有下面三个规定：

1) 变量名称的开头必须是一个英文字母，后面接的可以是英文字母、数字和下划线.

2) 区分大小写.

3) 变量名称不能超过 31 个.

此外，可以声明 MATLAB 的变量为局部变量或全局变量，如果一个函数内的变量没有特别声明，那么这个变量只在函数内部可用. 如果两个或多个函数想共用一个变量，那么就可以用 Global 来将它声明成全局变量.

全局变量的作用域是整个MATLAB工作空间，即全程有效. 所有的函数都可以对它们进行存取和修改，因此，定义全局变量是函数间传递数据的一个手段.

实际上，全局变量的使用可以减少参数传递. 如果一个系统能够合理地利用全局变量，将可以提高程序执行的效率. 另一方面，在做分散式系统时，全局变量的运用则可以使系统达到并行化的作用. 但是，值得注意：程序设计中，全局变量固然可带来某些方便，但却破坏了函数对变量的封装，降低了程序的可读性和可靠性. 因而，在结构化程序设计中，全局变量是不受欢迎的. 尤其当设计程序较大，子函数较多时，全局变量将给程序调试和维护带来不便，在这种情况下，故而不提倡使用全局变量；如果一定要用全局变量，那么最好给它起一个特别的名子，以避免和其他变量混淆.

例5.6　利用全局变量，建一个计算Fibonnaci函数的无参数传递的函数文件，并用它计算.

1）编写函数文件gfibno.m.

```
function f－gfibno
%gfibno. m
%利用全局变量ndcba计算Fibonnaci函数
global ndcba
f= [1, 1];
i=1;
while f (i) +f (i+1) <ndcba
      f (i+2) =f (i) +f (i+1);
      i=i+1;
end
```

2）在MATLAB指令窗口中，按如下(图5.8)方式运行.

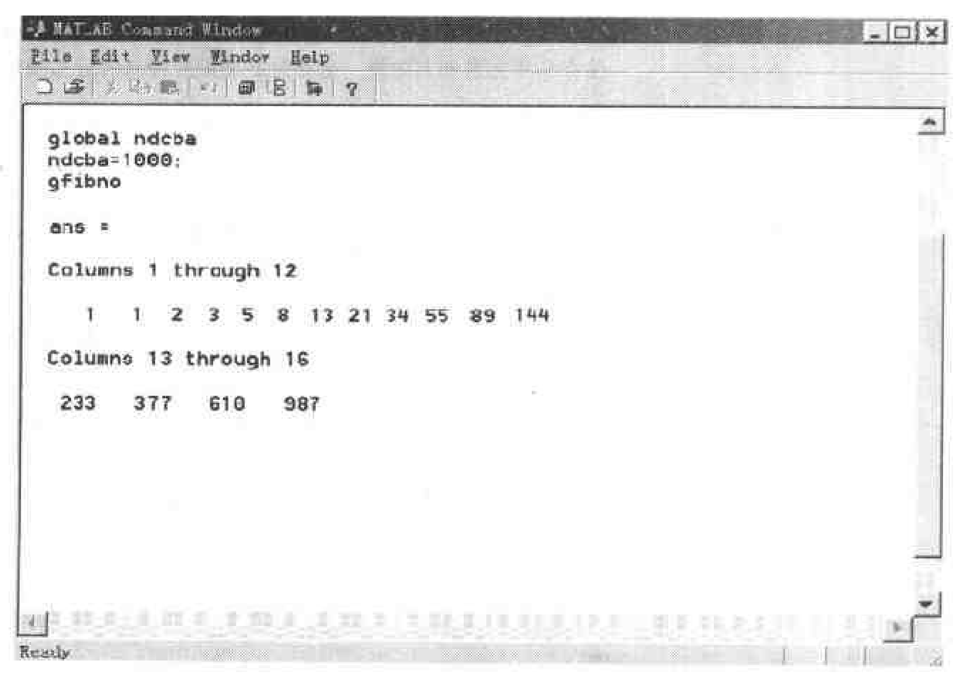


图5.8　例5.6程序的运行结果

5.3　数 据 类 型

MATLAB 里面定义了六种数据类型，即 char，double，sparse，uniy8，cell，struct，每种类型实际上都是数组格式的. 表 5.2 为这六种数据类型的详细说明，这些类型间的继承关系如图 5.9 所示. 一般较常用的只有两种：字母类型 char 和双精度浮点类型 double.

而大部分的函数也都提供了对这两种数据类型的处理. sparse 数据类型只是专门用在稀疏矩阵上，一方面可以节省内存空间，另一方面可以明显地提高运算速度，这类矩阵的例子如下：

$$\begin{bmatrix} 1 & 0 & 2 & 5.6 \\ 0 & 0 & 2.5 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0.2 \end{bmatrix},\quad \begin{bmatrix} 1 & 0 & \Lambda & 0 & 2 \\ 1 & 2 & 0 & \Lambda & 0 \\ 0 & \Lambda & 0 & \Lambda & 0 \\ 0 & \Lambda & 0 & n-1 & 0 \\ 1 & 2 & 3 & \Lambda & n \end{bmatrix}$$

假定给定一个矩阵 A

$$A=\begin{bmatrix} 0 & 0 & 1 \\ 0 & 15 & 1.2 \\ 0.34 & 0 & 0 \end{bmatrix}$$

表 5.2　各种数据类型的范例与说明

类　型	范　　例	说　　　明
double	[1.5 2; 0.01 0]	双精度数值数组，是 MATLAB 中最常用的数据类型. 一般的运算符、函数、数组函数都支持这种类型.
char	'Luoson-1'	字符数组，每一个字符用 16 位来表示.
sparse	speye (4)	双精度数值稀疏数组，当前只适用于二维数组. 稀疏数组中，只保存非零元素和其向量. 运算时必须配合特殊的运算函数，例如 splu，spchol.
cell	{'Luoson' 100 eye (2)}	单元数组. 可以包含其他不同类型数据的数组，适合大型数据库使用.
struct	Cat. name='mimi' Cat. age=1.2 Cat. color='yellow'	结构数组. 与单元数组一样，可以将不同的数据类型包含在同一个变量名称下，但结构数组另外含有数组名称.
unit8	Unit8 (magic (3))	无符号的 8 位整数数组. 数字范围 0～255. 当前尚无数学运算元可用，一般都用于数据保存或配合影像处理工具箱使用.
UserObject		用户定义的数据类型.

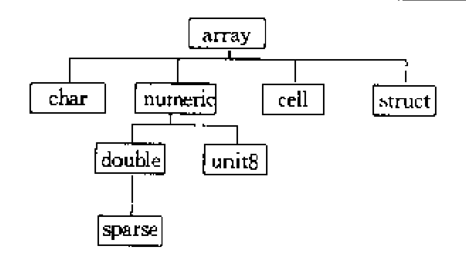


图 5.9　各种数据类型之间的继承关系

如果想将它保存为稀疏矩阵类型，可以通过如图 5.10 所示的命令进行处理.

```
MATLAB Command Window
File Edit View Window Help

a=[0 0 1;0 15 1.2;0.34 0 0]; %输入原矩阵
[i,j,s]=find(a);%将原矩阵非零元素，保存在s中
[m,n]=size(a);%求出矩阵的大小
s=sparse(i,j,s,m,n)%令s等于m行n列的稀疏矩阵

s =

  (3,1)       0.3400
  (2,2)      15.0000
  (1,3)       1.0000
  (2,3)       1.2000

Ready
```

图 5.10 将稀疏矩阵存成 sparse 的数据类型

5.4 程序结构

从理论上讲，只有顺序、循环和分支三种基本程序结构，就可构成任何一种程序并完成相应的工作. 与大多数计算机一样，MATLAB 有设计程序必须的程序结构：1）顺序结构；2）循环结构；3）分支结构.

为实现上述三种程序结构，MATLAB 提供了四种基本的流程控制方法，如表 5.3 所示.

表 5.3 四种基本的控制流程功能

关 键 字	功 能
if，else，elseif	根据逻辑条件执行一系列运算
switch，case，otherwise	根据条件值来选择执行的项目
while	根据逻辑条件来决定循环的执行次数
for	执行固定次数的循环

5.4.1 顺序结构

MATLAB 的顺序结构实际上就是复合表达式构成的语句. 复合表达式由分号或逗号隔离的几个表达式构成. 当表达式后面接分号时，表达式的计算结果虽不显示但中间结果仍保留在内存中. 若程序是命令文件，则程序运行完后，中间变量都予以保留；若程序是函数文件，那么程序运行完后，中间变量将被全部删除.

5.4.2　循环结构

在很多实际问题中会遇到许多规律的重复运算，因此在程序中就需要将某些语句重复执行．一组被重复执行的语句称为循环体，每循环一次，都必须作出是继续重复或是停止的决定，这个决定所依据的条件称为循环的终止条件，MATLAB 语言提供了两种循环方式：for-end 循环语句和 while-end 循环语句．

(1) for-end 循环语句

在许多情况下，循环条件是有规律变化的，通常是把循环条件的初值、判别和变化放在循环的开头，这种形式就是 for 循环结构．

for-end 循环语句的一般形式是：

```
for（计数器=初始值：增量：终止值）
    运算指令
end
```

该循环会依照计数器的值来决定运算指令的循环次数．其方法是：一开始计算器设定为初始值，并判断是否大于终止值，如果没有则执行运算指令；下一次将计数器加上增量，重复上次的判断直到计数器大于终止值时跳出循环．其中，如果不给定增量，MATLAB 会自动取为 1．在这个意义上，MATLAB 的 for 循环与其他计算机语言没有什么区别．

例 5.7　简单的 for 循环示例．

```
n=10;
for i=1: n
  x(i)=i;
end
x
x =
      1   2   3   4   5   6   7   8   9   10
```

例 5.8　循环的嵌套．

```
m=3;n=4;
for i=1:m
   for j=1:n
      a(i,j)=1/(i+j-1);
  end
end
format rat%设置输出格式，rat 为比值形式
a
a =
        1        1/2        1/3        1/4
        1/2      1/3        1/4        1/5
        1/3      1/4        1/5        1/6
```

for 循环语句的循环条件也可以是一个数组，例如 A 为 n×m 矩阵，则：

```
for (index=A)
  运算指令
end
```

在该例中，index 被设定为一维数组 A（:，k）．第一次循环中，k=1，然后反复执行，直到 k=m．换句话说，每次循环执行时，index 为 A 中一列的所有元素.

（2）while-end 循环语句

while 循环语句是使语句体在逻辑条件控制下重复不确定次，直到循环条件为假. while 循环的一般形式：

```
while (循环条件)
  运算指令
end
```

循环条件也可以是一个数组．如果该数组为空数组的话，MATLAB 会终止这个循环．例如：

```
while A
  运算指令
end
```

例 5.9　矩阵指数的幂级数展开为 $e^A=I+A+\frac{1}{2!}A^2+\frac{1}{3!}A^3+\cdots$．利用 while 循环语句，求矩阵指数．图 5.11 给出了其程序和运行结果.

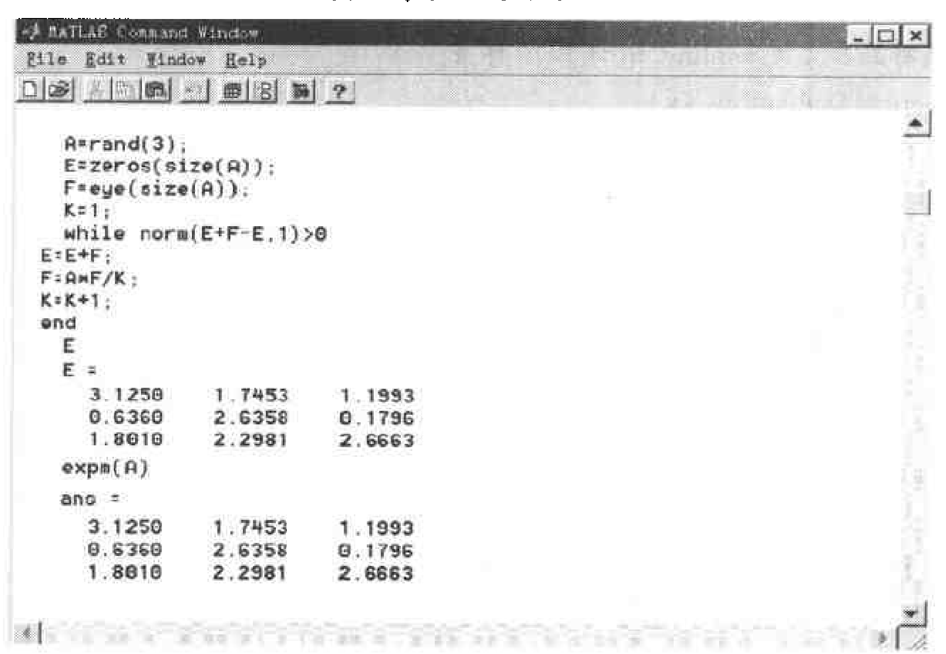


图 5.11　范例 5.9 运行结果

5.4.3　分支结构

在复杂的计算中常常需要根据表达式的情况（是否满足某些条件）确定下一步该做什么．MATLAB 的 if，else，elseif 与 switch 语句提供了描述条件分支的结构.

1. if，else，elseif 语句

if 用来检查逻辑运算、逻辑函数、逻辑变量值等逻辑表达式的真假，若为真则执行接下来的指令或运算，否则，转去执行另一分支. 写法如下：

```
if    逻辑表达式
   运算指令 1
else
   运算指令 2
end
```

在 MATLAB 中也可以利用 elseif 来写嵌套判断式，写法如下：

```
if    逻辑表达式 1
   运算指令 1
elseif 逻辑表达式 2
   运算指令 2
elseif 逻辑表达式 3
   运算指令 3
else
   运算指令 4
end
```

例 5.10　利用 MATLAB 的 if-else 语句实现多路选择.

1）编写函数文件 bspline.m 计算 B-样条.

```
function f=bspline (x)
if x<0
   f=0;
elseif x<1
   f=x;
elseif x<2
   f=2-x;
else
   f-0;
end
```

2）利用函数文件，进行实数分区间选择.

```
bspline (-1)
ans =
      0
bspline (1.6)
ans =
      0.4000
bspline (2)
```

```
ans =
     0
```

例5.11 解数论问题. 取任意整数，若是偶数，则用2除；否则乘3加1. 重复此过程，直到整数变为1. 有趣的是存在这样一个整数使问题无解，即循环将无止境的执行下去，能找到这个整数吗？

1）建立函数文件collatz.m.

```
function c=collatz(n)
%collatz
%Classic "3n+1"Problem from number theory
c=n;
while n>1
   if rem (n, 2) ==0
     n=n/2;
   else
      n=3*n+1;
   end
   c= [c, n];
end
```

2）利用函数文件，试探n=9所产生的情况.

```
collatz (9)
ans =
Columns 1 through 12
9  28  14  7  22  11  34  17  52  26  13  40
Columns 13 through 20
20  10  5  16  8  4  2  1
```

2. switch语句

switch语句也是MATLAB中的一个分支语句.如果在一个程序中,必须针对某个变量值来做多种不同的执行，switch比if-else更为方便. 此外，合理地使用switch语句也可以使程序更具有可读性.

switch的语法结构如下：

```
switch      分支条件（数值或字符串）
     case     数值（或字符串）条件1
                运算指令1
     case     数值（或字符串）条件2
                运算指令2
   otherwise
                运算指令n
end
```

基本的 switch 语句包含下列元素：

1）switch：switch 语句的开始，紧接着分支条件．分支条件可以是一个函数、变量或者表达式．

2）case：依照分支条件值，不同 case 可以定义不同的运算指令．而紧接在 case 后面的就是此 case 的分支条件．之后接着一个或一串运算指令．

3）otherwise：若不符合所有 case 的条件，则程序就会执行 otherwise 下面的表达式．

4）end：switch 语句的结束．

switch 实际上也是利用

if（分支条件＝＝数值条件）

比较来实现其目的，如果上面的判断为真，则符合条件，并且执行紧接的运算；反之，如判断的返回值为假，则继续检验下一个 case，直到最后一个失败，才执行 otherwise 下面的程序片段．如果分支条件是用于检测字符串条件的话，则其判断变成

if（strcmp（分支条件＝＝字符串条件））

switch 这个结构也经常用于一般的程序语言中，比如 C 语言．但如果你学过 C 语言，也许会有这样的疑问，即 MATLAB 的 switch 语句中为什么缺少了 break．其实很简单，在 C 语言中，程序执行所有符合条件的 case．换句话说，C 语言在检验某个 case 符合并执行其运算后，还会在继续检验下一个 case，直到全部检验完．所以，一般会在一个 case 描述的最后加入 break，让程序只运算第一个检验成功的运算式．MATLAB 在这方面则只执行第一个检验成功的 case．

5.5 程序流控制语句

有关程序流控制语句与函数，前面已零星地介绍过．本节将系统介绍 echo，input，pause，break 和 keyboard．

5.5.1 echo 指令

通常，m 文件中的指令不会显示在指令窗中．用 echo 命令可使文件指令在执行时可见，这对程序的调试和表演很有用．对命令文件和函数文件，echo 的作用稍微有些不同．

对命令文件，echo 较为简单，其格式为

echo on　　切换到显示其后所有被执行命令文件指令的状态．

echo off　　切换到其后所有被执行命令文件指令不被显示的状态．

echo　　实现被执行命令文件指令是否被显示的状态切换．

echo 的以上调用格式对函数文件不起作用．下面的 echo 调用格式对函数文件、命令文件都适用．具体如下：

echo FileName on　　使 FileName 指定文件的指令在执行中被显示出来．

echo FileName off　　终止显示 FileName 文件的执行过程．

echo FileName　　FileName 文件的执行过程是否被显示的切换开关．

echo on all　　显示其后所有被执行文件的过程．

echo off all　　使其后所有被执行文件的过程不被显示．

注意：当把echo运用于某一函数文件时，该文件将不被编译执行，而是被解释执行. 这样，函数文件在执行过程中，每一行都可被观察到. 由于这种解释执行不太有效，因而仅用于程序调试.

5.5.2　input，yesinput 指令

指令input提示用户从键盘输入数值、字符串或表达式，并接受该输入. 下面是几种常用的格式：

(1) a=input ('Please input a number:')

该指令运行后，将给出如下文字提示，并等待键盘输入：

Please input a number:

用户可以输入数字或表达式，也可以输入字符串（两端必须有单引号），按回车键确认后，该输入被赋给变量a.

(2) a=input ('Please input a number:','s')

该指令运行后，给出如下文字提示，并等待键盘输入：

Please input a number:

用户可以输入任何内容，（不管是数字还是字符）一律被当作字符串赋给变量a.

(3) a=yesinput ('Prompt:', Default, Possible)

yesinput是一个智能输入指令. 它带有缺省输入和输入的范围检查.

例5.12　带输入数值范围检查yesinput指令的用法.

1) 在MATLAB指令窗中键入

a=yesinput ('Order of the filter', 10, [0, 12]);

2) 该指令运行后，屏幕上提示如下：

Order of the filter (10):

3) 等待用户输入数据. 如果只是按回车键，则默认输入值为10；如果输入值大于12或小于0时，则认为输入无效，等待用户重新输入.

例5.13　带输入选项检查的yesinput指令的用法.

1) 在MATLAB指令窗中键入

color=yesinput ('color used on the plot','red','red|blue|green')

2) 该指令运行后，屏幕上提示如下：

color used on the plot (red):

3) 等待用户输入数据. 如果是按回车键，则默认输入值为red；如果输入的字符串与选项内容不符，则认为输入无效，等待用户重新输入.

5.5.3　pause 指令

pause指令使程序运行暂停，等待用户按任意键继续. pause命令在程序调试以及需要看中间结果时特别有用. pause的用法有两种：pause暂停执行程序；pause(n)在继续执行前，暂停n秒.

5.5.4 keyboard 指令

keyboard 与 input 一样有用. 当程序遇到 keyboard 指令时，MATLAB 将暂停程序的运行并调用你的机器的键盘命令进行处理. 一旦处理完自已的工作后，键人 return，然后按回车键，程序将继续进行. m 文件中含有它后，便于程序调试或在程序执行中修改变量.

5.5.5 break 指令

break 语句导致包含 break 指令的最内层 while，for，if，switch 语句的终止. 通过 break 语句，可不必等待循环的自然结束，而根据循环内部另设的某种条件是否满足，去决定是否退出循环，是否结束 if 等语句，在很多情况下这样做是十分必要的.

例 5.14 使用 break 求解两个自然数，这两个数的和等于 100，且第一个数被 2 整除的商与第二个数被 4 整除的商的和为 36.

```
i=1;
while 1
      if (rem (100-i*2, 4) ==0) & (i+ (100-i*2) /4) ==36
         break;
      end
     i=i+1;
end
a1=i*2
a2=100-i*2

a1 =
      44
a2 =
      56
```

5.6 函数调用及变量传递

MATLAB 中的函数调用及变量传递比较复杂. 而这两项工作又是编写高质量 m 文件所必不可少的. 对于一个较大的计算任务往往可以分成若干个比较小的任务，这就要碰到一个程序可以由若干个函数组成，并通过函数调用来完成.

5.6.1 函数调用

在 MATLAB 中，调用函数的常用形式是：

[输出参数 1，输出参数 2，…] = 函数名 (输人参数 1，输人参数 2，…)

注意：

1) 函数调用时各参数出现的顺序，应该与函数定义时的顺序一样，否则出错.

2）函数调用可以嵌套，一个函数可以调用别的函数，甚至调用它自已（即递归调用）.

例 5.15　给定两个实数 a，b 和一个正整数 n，求当 k=1，2，…，n 时的所有的 $(a+b)^n$ 与 $(a-b)^n$ 的值.

1）建立函数文件 ppower.m

```
function [out1,out2]=ppower(a,b,n)
%Ppower.m 计算(a+b)^n 和(a-b)^n
out1=(a+b)^n;
out2=(a-b)^n;
```

2)建立调用上述函数文件的命令文件 exp5.15.m

```
a=input('Pleas input a=:');
b=input('Pleas input b=:');
addpow=zeros(1,10);
subpow=zeros(1,10);
for k=1:10
    [addpow(k),subpow(k)]=ppower(a,b,k);
end
addpow
subpow
```

上述程序运行结果，如图 5.12 所示.

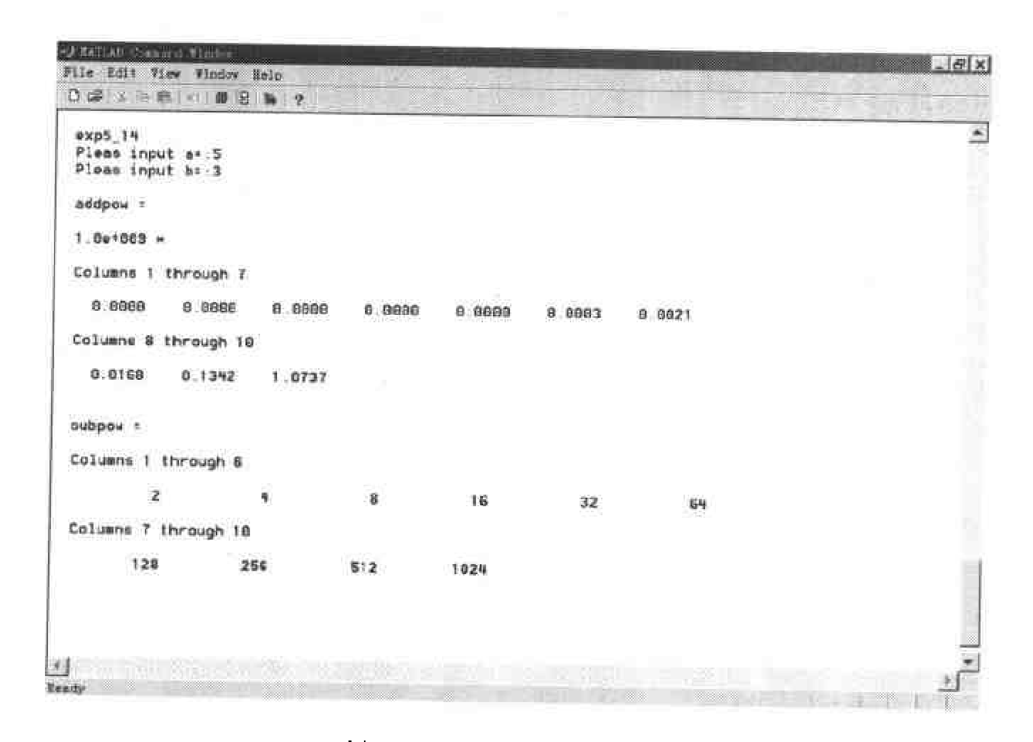


图 5.12　范例 5.15 的运行结果

说明：在上例里，exp5.15 对函数 power 每进行一调用，就传人三个参数 a，b，k，

送出两个结果 addpow 和 subpow.

5.6.2　参数传递

MATLAB 在函数调用上有一个与众不同之处：函数所传递参数的数目是可调的. 因此，它凭借这种特性，便可使一个函数可完成多种功能.

传递参数数目的可调性来源于如下两个 MATLAB 的永久变量：

nargin 函数体内的 nargin 给出调用该函数时的输入参数数目.

nargout 函数体内的 nargout 给出调用该函数时的输出参数数目.

只要在函数文件内包含这两个变量，就可准确地知道该函数文件被调用时的输入输出数，从而决定函数如何进行处理.

例 5.16　以 MATLAB 提供的“彗星线”绘制函数 comet. m 为例说明 nargin 的用法（该函数不带输出参数，但有三个可选的输入参数）.

```
function comet (x, y, p)
if nargin == 0, error('Not enough input arguments.'); end
if nargin < 2, y = x; x = 1:length(y); end
if nargin < 3, p = 0.10; end
ax = newplot;
if ~ishold,
axis([min(x(isfinite(x))) max(x(isfinite(x))) ...
min(y(isfinite(y))) max(y(isfinite(y)))])
end
co = get(ax,'colororder');
if size(co,1)>=3,
% Choose first three colors for head, body, and tail
head = line('color',co(1,:),'marker','o','erase','xor', ...
'xdata',x(1),'ydata',y(1));
body = line('color',co(2,:),'linestyle','-','erase','none', ...
'xdata',[],'ydata',[]);
tail = line('color',co(3,:),'linestyle','-','erase','none', ...
'xdata',[],'ydata',[]);
else
% Choose first three colors for head, body, and tail
head = line('color',co(1,:),'marker','o','erase','xor', ...
'xdata',x(1),'ydata',y(1));
body=line('color',co(1,:),'linestyle','--','erase','none', ...
'xdata',[],'ydata',[]);
tail = line('color',co(1,:),'linestyle','--','erase','none', ...
'xdata',[],'ydata',[]);
end
```

```
m = length(x);
k = round(p * m);
% Grow the body
for i = 2:k+1
j = i-1:i;
set(head,'xdata',x(i),'ydata',y(i))
set(body,'xdata',x(j),'ydata',y(j))
drawnow
end
% Primary loop
for i = k+2:m
j = i-1:i;
set(head,'xdata',x(i),'ydata',y(i))
set(body,'xdata',x(j),'ydata',y(j))
set(tail,'xdata',x(j-k),'ydata',y(j-k))
drawnow
end
% Clean up the tail
for i = m+1:m+k
j = i-1:i;
set(tail,'xdata',x(j-k),'ydata',y(j-k))
drawnow
end
```

5.7 神经网络应用设计举例

5.7.1 带有偏差单元的递归神经网络

尽管 BP 网络得到了广泛的应用，但其学习速率必须选得很小以保证学习过程的稳定性，这使得 BP 网学习过程很慢．因此，BP 网在很大程度上表现出它的不实用性，特别是对实时性很强的系统．

本节在 BP 网的基础上，加入反馈信号及偏差单元，生成了内部回归神经网络，由于这一网络结构上的特点，尤其是其在学习过程中便于引入经验知识（在偏差的选择上，可采用模糊知识概念），大大提高了学习速度．

内部回归神经网络（Internally Recurrent Net，IRN）就是利用网络的内部状态反馈来描述系统的非线性动力学行为．构成回归神经网络模型的方法有很多，但总的思想都是通过对前馈神经网络中加入一些附加的、内部的反馈通道来增加网络本身处理动态信息的能力．例如根据状态信息的反馈途径不同可构成两种不同的回归神经网络结构模型：Jordan 型和 Elman 型（如图 5.13）．

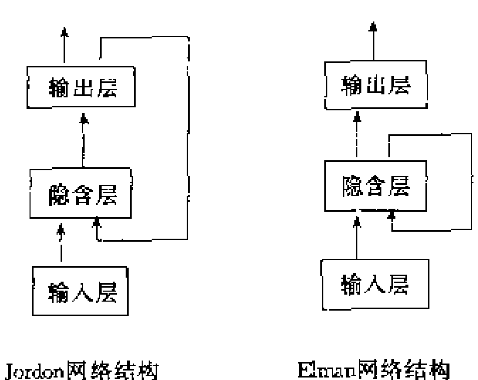


图 5.13　回归神经网络结构模型

本节首先针对多层 BP 网络的不足，在 Jordan 和 Elman 网络结构的基础上，给出一种带偏差单元的 IRN 网络模型及误差逆传播算法，最后应用带偏差单元的 IRN 网络，进行故障诊断方面的仿真分析.

1. BP 神经网络概述

典型的 BP 网络是三层网络，包括输入层、隐含层和输出层. 各层之间实行全连接，如图 5.14 所示.

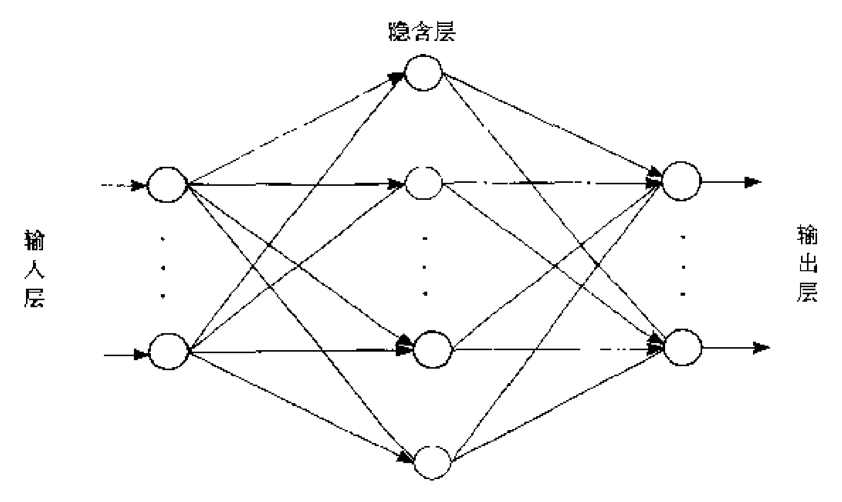


图 5.14　三层 BP 网络结构

BP 网络的学习由四个过程组成，即：输入模式由输入层经中间层向输出层的“模式顺传播”过程；网络的希望输出与网络实际输出之差的误差信号由输出层经中间层向输入层逐层修正连接权的“误差逆传播”过程；由“模式顺传播”与“误差逆传播”的反复交替进行的网络“记忆训练”过程；网络趋向收敛即网络的全局误差趋向极小值的“学习收敛”过程. 简言之，就是由“模式顺传播”→“误差逆传播”→“记忆训练”→“学习收敛”的过程. BP 网络学习规则有时也称广义 δ 规则.

2. BP 网络及算法的不足

比起早期的神经网络，BP 网络无论在网络理论还是网络性能方面都更加成熟. 其突出的优点就是具有很强的非线性映射能力和柔性的网络结构. 网络的中间层数、各层的

处理单元数及网络学习系数可根据具体情况任意设定，并且随着结构的差异其性能也有所不同.

但是，BP 网络并不是一个十分完善的网络，它存在以下一些主要缺陷：

1）学习收敛速度太慢，即使一个比较简单的问题，也需要几百次甚至上千次的学习才能收敛.

2）不能保证收敛到全局最小点.

3）网络隐含层的层数及隐含层的单元数的选取尚无理论上的指导，而是根据经验确定. 因此，网络往往有很大的冗余性，无形中也增加了网络学习的时间.

4）网络的学习、记忆具有不稳定性. 一个训练结束的 BP 网络，当给它提供新的记忆模式时，将使已有的连接权打乱，导致已记忆的学习模式的信息消失. 要避免这种现象，必须将原来的学习模式连同新加入的新学习模式一起重新进行训练. 而对于人类的大脑来说，新信息的记忆不会影响已记忆的信息，这就是人类大脑记忆的稳定性.

3. *带有偏差单元的递归神经网络*

图 5.15 给出了带有偏差单元的递归神经网络模型的结构，它由 3 层节点组成：输入层节点、隐层节点和输出节点，两个偏差节点分别被加在隐层和输出层上，隐层节点不仅接收来自输入层的输出信号，还接收隐层节点自身的一步延时输出信号，称为关联节点.

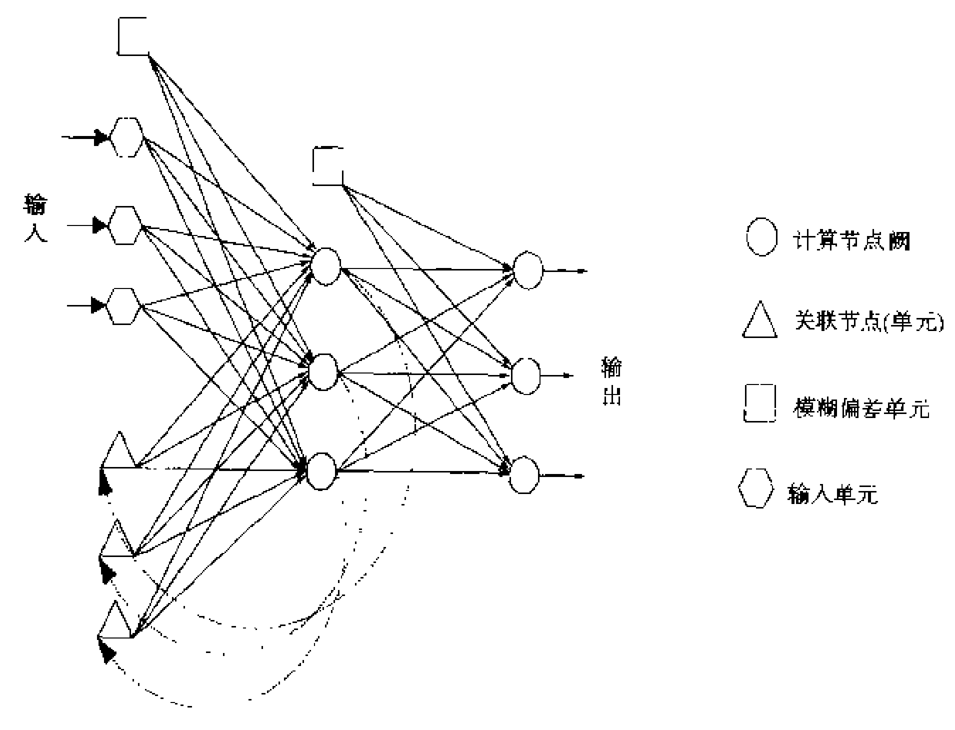


图 5.15　带有偏差单元的递归神经网络结构

设 NH 和 NI 分别为隐节点数和输入节点数（除偏差节点），$I_j(k)$是带有偏差单元的递归神经网络在时间 k 的第 j 个输入，$x_j(k)$是第 j 个隐层节点的输出，$y(k)$是带有偏差单元的递归神经网络的输出向量，则带有偏差单元的递归神经网络可由下列数学公式描述：

$$y(k) = \sum_{j=1}^{NH} WO_j x_j(k) + WO_{\text{bias}}$$
$$x_j(k) = \sigma(S_j(k)) \tag{5.1}$$
$$S_j(k) = \sum_{i=1}^{NH} WR_{ij} x_i(k-1) + \sum_{i=1}^{NI} WI_{ij} I_i(k) + WI_{j\text{bias}}$$

式中 $\sigma(\cdot)$ 是隐层节点的非线性激活函数，WI，WR，WO 分别为从输入层到隐层、回归信号、从隐层到输出层的权系数，WI_{bias}、WO_{bias}分别为加在隐层和输出层上的偏差单元的权系数. 由方程（5.1）可以看出，隐层节点的输出可以视为动态系统的状态，本章的 IRN 结构是非线性动态系统的状态空间表示. 带有偏差单元的递归神经网络的隐层节点能够存储过去的输入输出信息.

4. 带有偏差单元的递归神经网络的误差逆传播学习规则的数学推导

对照图 5.14 和图 5.15，带有偏差单元的递归神经网络同 BP 网络基本相近. 当带有偏差单元的递归神经网络的偏差单元和关联节点为 0 时，带有偏差单元的递归神经网络就是BP 网络，所以在考虑带有偏差单元的递归神经网络网络的权系数调整规则时，可以借用 BP 算法.

考虑图 5.14 所示三层 BP 网络，设输入模式向量 $A_k=(a_1, a_2, \cdots, a_n)$，希望输出向量 $Y_k=(y_1, y_2, \ldots, y_q)$；中间层单元输入向量 $S_k=(s_1, s_2, \ldots, s_p)$，输出向量 $B_k=(b_1, b_2, \ldots, b_p)$；输出层单元输入向量 $L_k=(l_1, l_2, \ldots, l_q)$，输出向量 $C_k=(c_1, c_2, \ldots, c_q)$；输入层至中间层连接权 $\{W_{ij}\}$，$i=1, 2, \ldots, n$，$j=1, 2, \ldots, p$；中间层至输出层连接权 $\{V_{jt}\}$，$j=1, 2, \ldots, p$，$t=1, 2, \ldots, q$；中间层各单元输出阈值为 $\{\theta_j\}$，$j=1, 2, \ldots, p$；输出层各单元输出阈值为 $\{\gamma_t\}$，$t=1, 2, \ldots, q$. 以上 $k=1, 2, \ldots, m$.

这里采用S 函数作为网络响应函数，它有一个重要特性，即：S 函数的导数可用S 函数自身表示，如式(5.2)和(5.3)所示.

$$f(x) = \frac{1}{1+e^{-x}} \tag{5.2}$$

$$f'(x) = f(x)[1 - f(x)] \tag{5.3}$$

设第 k 个学习模式网络希望输出与实际输出的偏差为

$$\delta_j^k = (y_j^k - C_j^k),\ j = 1,2,\cdots,q$$

δ_j^k 的均方值为

$$E_k = \sum_{t=1}^{q} (y_t^k - C_t^k)^2/2 = \sum_{t=1}^{q} (\delta_t^k)^2/2 \tag{5.4}$$

为使 E_k 随连接权的修正按梯度下降，则需求 E_k 对网络实际输出 $\{C_t^k\}$ 的偏导

$$\frac{\partial E_k}{\partial C_t^k} = -(y_t^k - C_t^k) = -\delta_t^k \tag{5.5}$$

由于

$$L_t = \sum_{j=1}^{p} V_{jt} b_j - \gamma_t,\ t = 1,2,\cdots,q \tag{5.6}$$

$$C_t^k = f(L_t),t = 1,2,\cdots,q \tag{5.7}$$

连接权 V_{jt}的微小变化对输出层响应的影响，可由式(5.3)和(5.6)推得式(5.8)

$$\frac{\partial C_t^k}{\partial V_{jt}} = \frac{\partial C_t^k}{\partial L_t} \cdot \frac{\partial L_t}{\partial V_{jt}} = f'(L_t) \cdot b_j = C_t^k(1 - C_t^k) \cdot b_j \tag{5.8}$$

$$t = 1,2,\cdots,q, \qquad j = 1,2,\cdots,p$$

则连接权 V_{jt} 的微小变化对第 k 个模式的均方差 E_k 的影响，可由式 (5.5) 和 (5.8) 推得

$$\frac{\partial E_k}{\partial V_{jt}} = \frac{\partial E_k}{\partial C_t^k} \cdot \frac{\partial C_t^k}{\partial V_{jt}} = -\delta_t^k C_t^k (1 - C_t^k) \cdot b_j \tag{5.9}$$

$$t = 1,2,\cdots,q, \qquad j = 1,2,\cdots,p$$

按梯度下降原则，使连接权 V_{jt} 的调整量 ΔV_{jt} 与 $\frac{\partial E_k}{\partial V_{jt}}$ 的负值成比例变化，则由式 (5.9) 可得

$$\Delta V_{jt} = -\alpha \left[\frac{\partial E_k}{\partial V_{jt}} \right] = \alpha \delta_t^k C_t^k (1 - C_t^k) \cdot b_j \tag{5.10}$$

$$0 < \alpha < 1, t = 1,2,\cdots,q, j = 1,2,\cdots,p$$

设输出层各单元的一般化误差为 d_t^k，$t=1, 2, \cdots, q$；$k=1, 2, \cdots, m$. d_t^k 定义为 E_k 对输出层输入 L_t 的负偏导，由式 (5.5) 和 (5.3) 可得

$$d_t^k = -\frac{\partial E_k}{\partial L_t} = -\frac{\partial E_k}{\partial C_t^k} \cdot \frac{\partial C_t^k}{\partial L_t} = \delta_t^k C_t^k (1 - C_t^k) \tag{5.11}$$

$$t = 1,2,\cdots,q, \qquad k = 1,2,\cdots,m$$

则连接权 V_{jt} 的调整量 ΔV_{jt} 可表示为

$$\Delta V_{jt} = \alpha \cdot d_t^k \cdot b_j \tag{5.12}$$

$$t = 1,2,\cdots,q, j = 1,2,\cdots,p, k = 1,2,\cdots,m$$

同理，由输入层至中间层连接权的调整，仍然按梯度下降法的原则进行. 中间层各单元的输入 $\{S_j\}$ 为

$$S_j = \sum_{i=1}^{n} W_{ij} \cdot a_j - \theta_j, \qquad j = 1,2,\cdots,p \tag{5.13}$$

其输出 $\{b_j\}$ 为

$$b_j = f(S_j), \qquad j = 1,2,\cdots,p \tag{5.14}$$

连接权 W_{ij} 的微小变化，对第 k 个学习模式的均方误差的影响，可由式 (5.11)、(5.6)、(5.14)、(5.3)、(5.13) 推得式 (5.15)

$$\begin{aligned}\frac{\partial E_k}{\partial W_{ij}} &= \left[\sum_{t=1}^{q} \frac{\partial E_k}{\partial L_t} \cdot \frac{\partial L_t}{\partial b_j} \right] \cdot \frac{\partial b_j}{\partial S_j} \cdot \frac{\partial S_j}{\partial W_{ij}} = \left[\sum_{t=1}^{q} (-d_t^k) V_{jt} \right] \cdot f'(S_j) \cdot a_i \\ &= -\left[\sum_{t=1}^{q} d_t^k V_{jt} \right] \cdot b_j \cdot (1 - b_j) \cdot a_i\end{aligned} \tag{5.15}$$

$$i = 1,2,\cdots,n, \qquad j = 1,2,\cdots,p$$

设中间层各单元的一般化误差为 $\{e_j^k\}$，$j=1, 2, \cdots, p$；$k=1, 2, \cdots, m$. e_j^k 定义为 E_k 对中间层输入 S_j 的负偏导. 由式 (5.11)、(5.6)、(5.14) 和 (5.3) 可得

$$e_j^k = -\frac{\partial E_k}{\partial S_j} = -\left[\sum_{t} \frac{\partial E_k}{\partial L_t} \cdot \frac{\partial L_t}{\partial b_j} \right] \cdot \frac{\partial b_j}{\partial S_j} = \left[\sum_{t=1}^{q} d_t^k \cdot V_{jt} \right] \cdot b_j \cdot (1 - b_j) \tag{5.16}$$

$$j = 1,2,\cdots,p, \qquad k = 1,2,\cdots,m$$

则式(5.15)可表示为

$$\frac{\partial E_k}{\partial W_{ij}} = -e_j^k \cdot a_i \tag{5.17}$$

$$i = 1,2,\cdots,n, \qquad j = 1,2,\cdots,p$$

与式（5.10）类似，连接权 W_{ij} 的调整量应为

$$\Delta W_{ij} = -\beta \frac{\partial E_k}{\partial W_{ij}} = \beta \cdot e_j^k \cdot a_i \tag{5.18}$$

$$i = 1,2,\cdots,n, \qquad j = 1,2,\cdots,p$$

同理阈值 $\{\gamma_t\}$、$\{\theta_j\}$ 的调整量为

$$\Delta\gamma_t = \alpha \cdot d_t^k, \qquad t = 1,2,\cdots,q \tag{5.19}$$

$$\Delta\theta_j = \beta \cdot e_j^k, \qquad j = 1,2,\cdots,p \tag{5.20}$$

以上的推导仅是针对一组学习模式进行的．设网络的全局误差为 E，则

$$E = \sum_{k=1}^{m} E_k = \sum_{k=1}^{m}\sum_{t=1}^{q}(y_t^k - C_t^k)^2/2 \tag{5.21}$$

从以上的推导可以看出，各个连接权的调整量是分别与各个学习模式对的误差函数 E_k 成比例变化的，称为标准误差逆传播算法．而相对于全局误差函数 E 的连接权的调整，应该在所有 m 个学习模式全部提供给网络之后统一进行，称为累积误差逆传播算法．

下面给出整个学习过程的具体步骤和流程图：

1）初始化．

2）选取模式对 A_k，Y_k 提供给网络．

3）用输入模式 A_k，连接权 $\{W_{ij}\}$ 计算中间层各单元的输入 S_j，然后用 $\{S_j\}$ 通过 S 函数计算中间层各单元的输出 $\{b_j\}$

$$S_j = \sum_{i=1}^{n} W_{ij} \cdot a_i - \theta_j, \qquad j = 1,2,\cdots,p$$

$$b_j = f(S_j), \qquad j = 1,2,\cdots,p$$

4）用中间层的输出 $\{b_j\}$、连接权 $\{V_{jt}\}$ 计算输出层各单元的输入 $\{L_t\}$，然后用 $\{L_t\}$ 通过 S 函数计算输出层各单元的响应 $\{C_t^k\}$

$$L_t = \sum_{j=1}^{p} V_{jt} \cdot b_j - \gamma_t, t = 1,2,\cdots,q$$

$$C_t^k = f(L_t), t = 1,2,\cdots,q$$

5）用希望输出模式 Y_k、网络实际输出 $\{C_t^k\}$ 计算输出层的各单元的一般化误差 $\{d_t^k\}$

$$d_t^k = (y_t^k - C_t^k) \cdot C_t^k \cdot (1 - C_t^k), \qquad t = 1,2,\cdots,q$$

6）用连接权 $\{V_{jt}\}$、输出层的一般化误差 $\{d_t^k\}$、中间层的输出 $\{b_j\}$ 计算中间层各单元的一般化误差 $\{e_j^k\}$

$$e_j^k = \left[\sum_{t=1}^{q} d_t^k \cdot V_{jt}\right] \cdot b_j \cdot (1 - b_j), \qquad j = 1,2,\cdots,p$$

7）用输出层各单元的一般化误差 $\{d_t^k\}$、中间层各单元的输出 $\{b_j\}$ 修正连接权 $\{V_{jt}\}$

$$V_{jt}(N+1) = V_{jt}(N) + \alpha \cdot d_t^k \cdot b_j$$

$$j=1,2,\cdots,p,\quad t=1,2,\cdots,q,\quad 0<\alpha<1$$

8）用中间层各单元的一般化误差 $\{e_j^k\}$、输入层各单元的输入 A_k 修正连接权 $\{W_{ij}\}$

$$W_{ij}(N+1)=W_{ij}(N)+\beta\cdot e_j^k\cdot a_i^k$$

$$i=1,2,\cdots,n,\quad j=1,2,\cdots,p,(0<\beta<1)$$

9）选取下一个学习模式对提供给网络，返回到步骤 3)，直到全部 m 个模式对训练完毕.

10）重新从 m 个学习模式对中随机选取一个模式对，返回步骤 3)，直至网络全局误差函数 E 小于预先设定的一个极小值.

11）结束学习.

学习过程的流程图如图 5.16 所示.

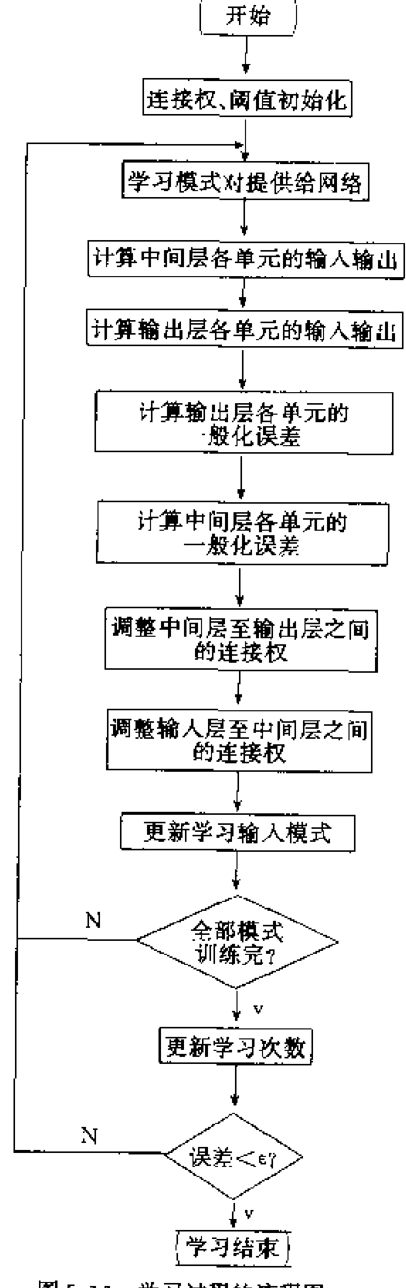


图 5.16　学习过程的流程图

5. 带有偏差单元的递归神经网络诊断模型的建立

近几年以来，回归神经网络的研究越来越受重视. 其应用领域不断扩大，例如 Su 于 1992 年成功地应用回归神经网络对非线性系统进行建模，Ku&Lee，Narendra 在非线性系统辨识和控制中采用了 IRN 模型，获得了满意的效果.

已发展起来的神经网络故障诊断模型，主要包括三层（BP 网）：1）输入层，即从实际系统接收的各种故障信息及现象. 2）中间层，是把从输入层得到的故障信息，经内部的学习和处理，转化为针对性的解决办法. 3）输出层，是针对输入的故障形式，经过调整权系数 W_{ij}后，得到的处理故障方法. 简而言之，神经网络模型的故障诊断就是利用样本训练收敛稳定后的节点连接权值，向网络输入待诊断的样本征兆参数，计算网络的实际输出值，根据输出值的大小排序，从而确定故障类别. 图 5.17 表示基于神经网络的故障分类诊断的一般流程图.

测点信息 → 预处理 → 特征提取 → 神经网络 → 诊断结果

图 5.17 神经网络故障诊断流图

下面，用带有偏差单元的递归神经网络来实现故障分类. 带有偏差单元的递归神经网络输入层有 5 个神经元对应 5 个测试点，输出层有 5 个神经元，隐层有 10 个神经元，其他关联节点和偏差单元的结构配置与图 5.15 相类似.

训练样本如表 5.4 所示，以测试编码作为网络输入，以故障编码作为网络输出，第一层学习率为 1.5，第二层学习率为 1.5，输入偏差学习率为 1.0，输出偏差学习率为 3000，网络学习到第 7 步，其精度优于 0.01，图 5.18 为带有偏差单元的递归神经网络误差的收敛结果.

将训练好的网络冻结，以测试编码为输入，使网络处于回想状态，回想结果如表 5.5 所示.

表 5.4 故障编码

故障序号	测试编码	故障编码
1	11111	00000
2	01000	10000
3	10000	01000
4	11000	00100
5	11100	00010
6	11110	00001

其程序清单如下：

```
clear
%标准输入输出数据
p= [1 1 1 1 1
    0 1 0 0 0
    1 0 0 0 0
    1 1 0 0 0
    1 1 1 0 0
```

```
      1 1 1 1 0];
t= [0 0 0 0 0
    1 0 0 0 0
    0 1 0 0 0
    0 0 1 0 0
    0 0 0 1 0
    0 0 0 0 1];
```

表 5.5 网络对训练模式的回想结果

测试编码					
11111	01000	10000	11000	11100	11110

故障编码				
bit1	bit2	bit3	bit4	bit5
0.0000	0.0001	0.0000	0.0000	0.0000
0.9922	0.0000	0.0002	0.0001	0.0001
0.0000	0.99922	0.0002	0.0001	0.0001
0.0000	0.0000	0.9999	0.0002	0.0001
0.0001	0.0001	0.0000	0.9952	0.0001
0.0001	0.0001	0.0000	0.0000	0.9985

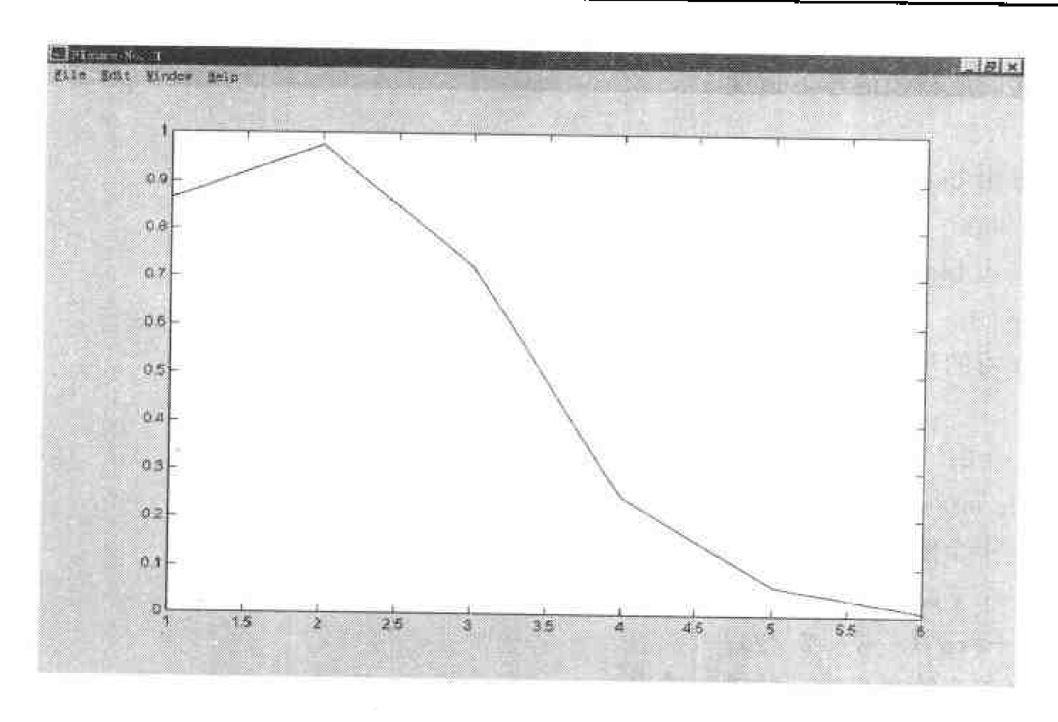


图 5.18 训练误差曲线

```
%给权值赋初值
w1=eye(5,10);
w2=eye(10,5);
wr=eye(10,10)/3;
wobias=eye(6,5)/4;
```

```
wbias=eye(6,10)/6;
x=ones(6,10)/3;
ww2=zeros(10,5)/6;
ww1=zeros(5,10)/6;
wwr=zeros(10,10)/6;
wwobias=zeros(6,5)/5;
wwbias =zeros(6,10)/4;
g = [1 1 1 1 1];
f = [1 1 1 1 1 1 1 1 1 1] ;
mmax=0.2;
mmmax=0.1;
%要求的偏差值
h=0.04;
u=0.04;
%输出层权值的学习速度
a =1.5
%隐含层权值的学习速度
b =1.18
%递归层权值的学习速度
v =1.5;
%输出 bias unit 的学习速度
r =3000;
%输入 bias unit 的学习速度
w =10;
%学习的步数
n =0;
mm =0;
while mmax>0.01
  %十个隐含层单元的输入输出
  s=p * w1+x * wr+h * wbias;
  x=exp (-s. ^ 2./2);
  %五个输出层单元的输入输出
        y=x * w2+u * wobias;
        c=exp(-y. ^ 2./2);
        %希望的输出与实际的输出的偏差
        j=t-c;
        dj=max(abs(j));
        mmax=max(dj')
          if mmax>0.04
```

```
        for k=1:6
          %输出层单元的一般化误差
          d =-j. * y. * exp(-y. ^ 2. /2);
          %隐含层单元的一般化误差
          e = - d * w2'. * s. * exp(-s. ^ 2. /2);
          ww2=ww2+a * (f' * d(k,:)). * (g' * x(k,:))';
          wwobias=wwobias+r * d * h;
          ww1=ww1+b * (f' * p(k,:))'. * (g' * e(k,:));
          wwr=wwr+v * (f' * x(k,:))'. * (f' * e(k,:));
          wwbias=wwbias+w * e * u;
            end
            ww2=ww2. /6;
            ww1=ww1. /6;
            wwr=wwr. /6;
            wwobias=wwobias. /6;
            wwbias=wwbias. /6;
            w2=w2+ww2;
            w1=w1+ww1;
            wr=wr+wwr;
            wobias=wobias+wwobias;
            wbias=wbias+wwbias;
            end
      mm=mm+1;
      n=n+1
      nn(mm)=n;
      ee(n) =mmax;
      ww2=zeros(10,5)/6;
      ww1=zeros(5,10)/6;
      wwr=zeros(10,10)/6;
      wwobias=zeros(6,5)/5;
      wwbias =zeros(6,10)/4;
  end
%找出所有实际输出与希望输出的最大误差
%所有模式训练后的满足要求的实际输出
c
x=1:1:n
plot(x,ee)
%xlabel('训 练 步 数')
%ylabel('最大误差')
```

5.7.2 具有快速学习算法的补偿模糊神经网络

模糊理论与神经网络技术的研究不仅在各自的学科里取得了引人注目的进展，而且在这两大学科的边缘开辟了众多的研究新领域，二者的相互渗透和有机结合，促进了模糊神经网络技术的研究.

常规的模糊神经网络在设计过程中往往存在技术上的难点，例如，如何优选模糊隶属函数；如何优选模糊逻辑推理和最优模糊运算；如何优选反模糊函数等. 为了克服这些困难，本节在融合模糊理论和神经网络技术的基础上，介绍了一种具有快速学习算法、能够执行补偿模糊推理的补偿模糊神经网络，进行了函数逼近的仿真分析，并且将其成功运用于未建模系统的故障检测中.

1. 模糊逻辑系统的特点

模糊逻辑系统由模糊产生器、知识库、模糊推理机和反模糊化器四部分组成，如图 5.19 所示.

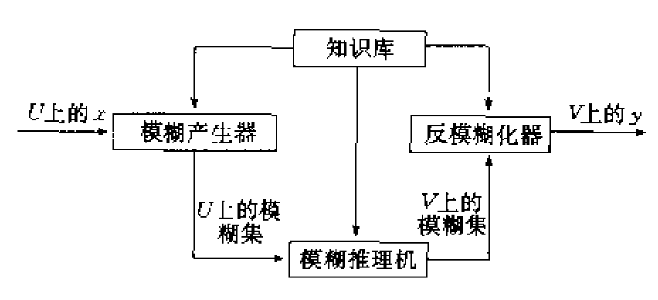


图 5.19 模糊逻辑系统

模糊产生器将论域 U 上的点一一映射为 U 上的模糊集合. 论域 $U=\{x\}$ 上的模糊集合是指 x 中具有某种性质的元素全体，这些元素具有某个不分明的界限，对于 U 中任一元素，都能根据这种性质，用一个 [0, 1] 上的隶属函数 $\mu_A(x)$ 来表征该元素属于 A 的程度. $\mu_A(x)$ 的大小反映了 x 对于模糊集合 A 的隶属程度；模糊推理机根据模糊规则库中的模糊推理知识以及由模糊产生器产生的模糊集合，基于模糊逻辑中的蕴涵关系及推理规则，推理出模糊结论，即论域 V 上的模糊集，并将其输入到反模糊化器；反模糊化器将论域 V 上的模糊集合一一映射为 V 上确定的点.

2. 模糊理论与神经网络技术的融合

模糊逻辑模仿人脑的逻辑思维，用于处理模型未知或不精确的控制问题；神经网络模仿人脑神经元的功能，可作为一般的函数估计器，映射输入输出关系. 二者的结合实际是人类大脑结构和功能的模拟，即大脑神经网络“硬件”拓扑结构和信息模糊处理“软件”思维功能的结合. 随着模糊神经网络技术的发展，模糊系统和神经网络的融合方式为构造各类模糊神经元及模糊神经网络，作为模糊信息处理单元以实现模糊信息的自动化处理. 主要体现在四个方面：模糊系统和神经网络的简单结合、用模糊理论增强的神经网络、用神经网络增强的模糊系统和借鉴模糊系统设计的神经网络结构.

3. 补偿模糊神经网络

补偿模糊神经网络是一个结合了补偿模糊逻辑和神经网络的混合系统，由面向控制和面向决策的模糊神经元所构成，这些模糊神经元被定义为执行模糊化运算、模糊推理、补偿模糊运算和反模糊化运算. 由于补偿模糊逻辑神经网络引入了补偿模糊神经元，使网络能够从初始正确定义的模糊规则或者初始错误定义的模糊规则进行训练，使网络容错性更高，系统更稳定；同时，常规的模糊神经网络中，模糊运算往往采用静态的、局部优化运算方法，例如，最小运算、最大运算、乘积运算或者代数和运算. 而补偿模糊神经网络中，模糊运算采用了动态的、全局优化运算，并且在神经网络的学习算法中，又动态地优化了补偿模糊运算，使网络更适应、更优化. 网络不仅能适当调整输入、输出模糊隶属函数，也能借助于补偿逻辑算法动态地优化适应的模糊推理，其网络参数具有明确的物理含义，可以用一个启发式算法去预置，加快训练速度.

（1）模糊神经元

通常，一个执行模糊化运算的模糊神经元被称为模糊化神经元；一个执行模糊推理的模糊神经元被称为模糊推理神经元；一个执行反模糊化运算的模糊神经元被称为反模糊化神经元. 一个执行补偿模糊运算的模糊神经元被称为补偿模糊神经元.

1）模糊化神经元. 一个典型的模糊化神经元如图 5.20 所示. 其中，Ai 是模糊语言变量 A 的模糊语言项，模糊化神经元执行一个从确定值 x 到模糊子集 Ai 的映射，用 $y=\mu_{A_i}(x)$ 表示.

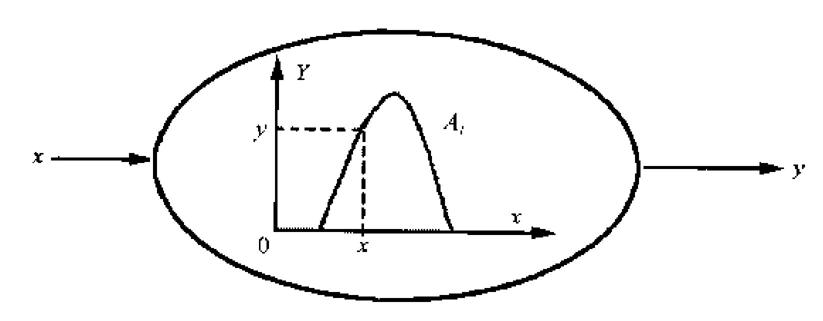


图 5.20 模糊化神经元

2）模糊推理神经元. 一个简单的模糊推理神经元如图 5.21 所示. 它能通过一些函数 T（x_1、x_2，…，x_n）映射输入 x_i（$i=1$，2，…，n）到输出 y.

3）反模糊化神经元. 一个典型的反模糊化神经元如图 5.22 所示.

它能产生基于输入 x_i（$i=1$，2，…，n）和权值 w_i^k（$i=1$，2，…，n；$k=1$，2，…，m）的确定值 y，其中，权值 w_i^k 是输出隶属函数的参数. 一个典型的反模糊化函数如式(5.22)所示.

$$D(x_1,x_2,\cdots,x_n)=\frac{\sum_{i=1}^{n} w_i^1 w_i^2 x_i}{\sum_{i=1}^{n} w_i^2 x_i} \tag{5.22}$$

具有可调参数 w_i^1 和 $w_i^2(i=1,2,\cdots,n)$ 的输出隶属函数如式(5.23)所示.

$$\mu_{Y_i}(y)=e^{\{-[(y-w_i^1)/W_i^2]^2\}} \tag{5.23}$$

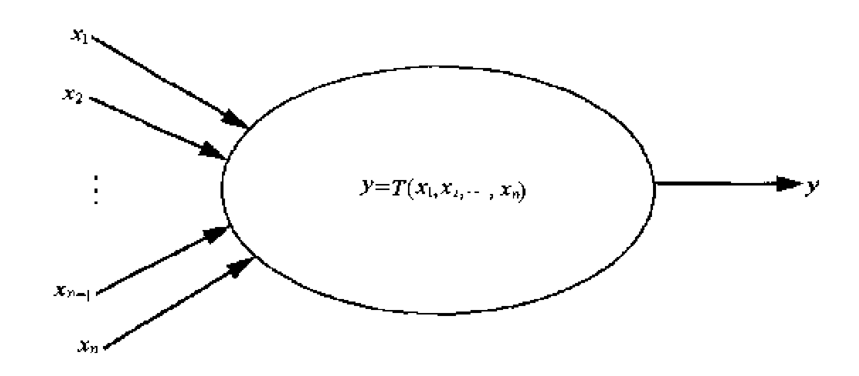


图 5.21　模糊推理神经元

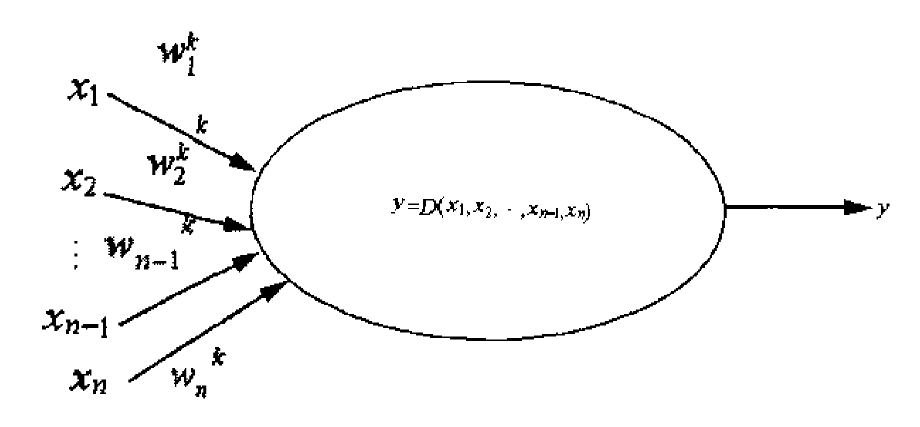


图 5.22　反模糊化神经元

其中 w_i^1 和 w_i^2 分别是输出隶属函数的中心和宽度.

4）补偿模糊神经元. 根据补偿运算的实质，提出了一种基于消极运算和积极运算的补偿运算.

a. 消极模糊神经元. 消极模糊神经元如图 5.23 所示. 它能够映射输入 x_i（$i=1$，2，…，n）到最坏的输出，为最坏的情形制定一个保守的决策，例如，x_i（$i=1$，2，…，n）在 [0，1] 中，$P(x_1, x_2, \cdots, x_n)=\min(x_1, x_2, \cdots, x_n)$.

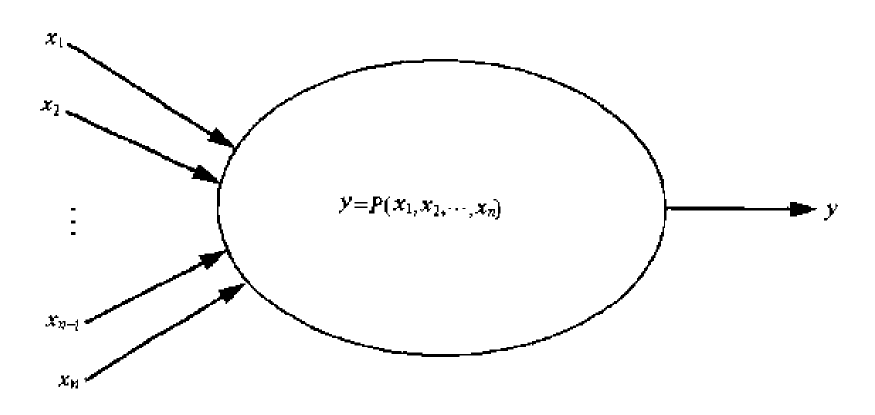


图 5.23　消极模糊神经元

b. 积极模糊神经元. 积极模糊神经元如图 5.24 所示. 它能够映射输入 x_i（$i=1$，2，…，n）到最好的输出，为最好的情形制定一个乐观的决策，例如，x_i（$i=1$，2，…，n）

在 [0, 1] 中. $O(x_1, x_2, \cdots, x_n) = \max(x_1, x_2, \cdots, x_n)$.

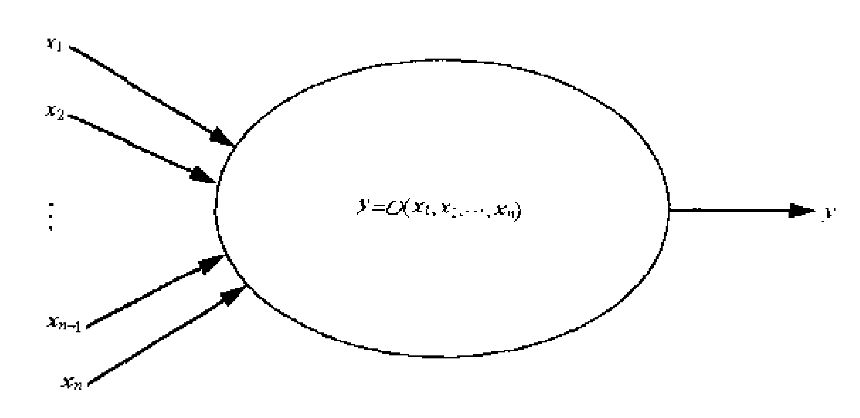


图 5.24 积极模糊神经元

c. 补偿模糊神经元. 补偿模糊神经元如图 5.25 所示.

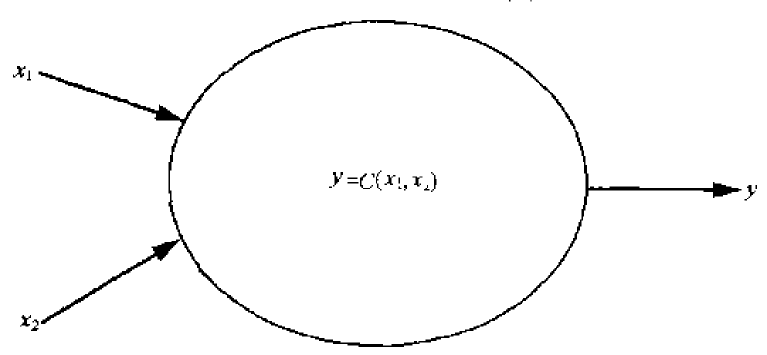


图 5.25 补偿模糊神经元

它能够映射最坏输入 x_1 和最好输入 x_2 到补偿输出 y,并为界于最坏输入和最好输入的情况制定一个相对折衷的决策. 例如:

$$C(x_1, x_2) = x_1^{1-\gamma} x_2^{\gamma} \tag{5.24}$$

其中 $\gamma \in [0, 1]$ 为补偿度.

(2) 补偿模糊神经网络结构

一个补偿模糊神经网络有五层结构:输入层、模糊化层、模糊推理层、补偿运算层、反模糊化层. 层与层之间依据模糊逻辑系统的语言变量、模糊 IF-THEN 规则、最坏-最好运算、模糊推理方法、反模糊函数所构建, 其结构如图 5.26 所示. 其中第一层的各个结点直接与输入向量相连接;第二层的每一个结点代表一个语言变量值,其作用是计算各输入向量属于各语言变量值模糊集合的隶属函数;第三层的每一个结点代表一条模糊规则,其作用是匹配模糊规则,并计算出每条规则的适用度.

(3) 补偿模糊推理

具有 N 输入 1 输出的补偿模糊逻辑系统的 M 条模糊 IF-THEN 规则表述如下:

FR$^{(k)}$: IF x_1 is A_1^k and ... and x_n is A_n^k

THEN y is B^k

其中 A_i^k 是论域 U 上的模糊集, B^k 是论域 V 上的模糊集. x_i 和 y 是语言变量, $i=1, 2, \cdots, n$; $k=1, 2, \cdots, m$.

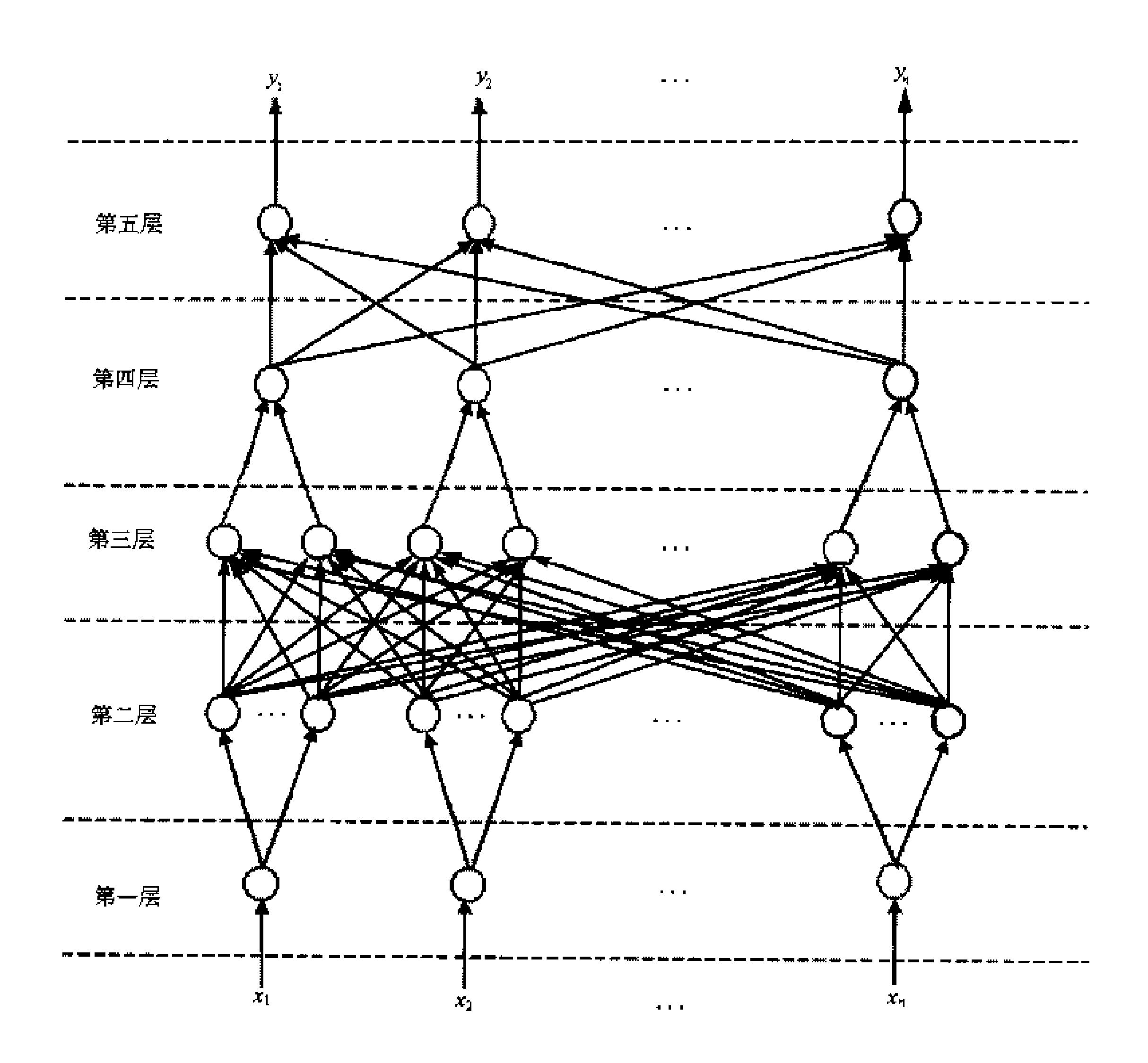


图 5.26　补偿模糊神经网络

模糊隶属函数为

$$\mu_{A_i^k}(x) = \exp\left[-\left(\frac{x_i - a_i^k}{\sigma_i^k}\right)^2\right] \tag{5.25}$$

$$\mu_{B^k}(y) = \exp\left[-\left(\frac{y - b^k}{\delta^k}\right)^2\right] \tag{5.26}$$

其中 a 与 σ 为输入隶属函数的中心和宽度；b 与 δ 为输出隶属函数的中心和宽度.

定义输入$\underline{X}=(x_1, \cdots, x_n)$，论域为 $U=U1\times U2\cdots\times Un$，对于论域 U 中一个输入模糊子集 A'，根据第 k 个模糊规则，能够在输出论域 V 中产生一个输出模糊子集 B'. 模糊推理采用最大-代数积（sup-·）合成运算，则由模糊推理规则所导出的 V 上的模糊集合 B' 为

$$\mu_{B'^k}(y) = \sup_{\underline{x}\in U}(\mu_{A_1^k\times\cdots\times A_n^k\to B^k}(\underline{x}, y)\cdot\mu_{A'}(\underline{x})) \tag{5.27}$$

模糊蕴涵采用积运算（Larsen）

$$R_p = A \to B \text{ 即 } \mu_{A\to B}(x, y) = \mu_A(x)\mu_B(y) \tag{5.28}$$

$$\mu_A(x) \cdot \mu_B(y) = \mu_A(x)\mu_B(y) \tag{5.29}$$

消极运算为

$$u^k = \prod_{i=1}^{n} \mu_{A_i^k}(x_i) \tag{5.30}$$

积极运算为

$$v^k = \left[\prod_{i=1}^{n} \mu_{A_i^k}(x_i)\right]^{1/n} \tag{5.31}$$

补偿运算为

$$\mu_{A_1^k \times \cdots \times A_n^k}(\underline{x}) = (u^k)^{1-\gamma}(v^k)^\gamma = \left[\prod_{i=1}^{n} \mu_{A_i^k}(x_i)\right]^{1-\gamma+\gamma/n} \tag{5.32}$$

其中 γ 为补偿度，$\gamma \in [0,1]$，因此

$$\mu_{B'^k}(y) = \sup_{\underline{x} \in U}\left\{\mu_{B^k}(y)\mu_{A'}(\underline{x})\left[\prod_{i=1}^{n} \mu_{A_i^k}(x_i)\right]^{1-\gamma+\gamma/n}\right\} \tag{5.33}$$

采用单值模糊化，$\mu_{A'}(\underline{x})=1$，$\mu_{B^k}(b^k)=1$，则

$$\mu_{B'^k}(b^k) = \left[\prod_{i=1}^{n} \mu_{A_i^k}(x_i)\right]^{1-\gamma-\gamma/n} \tag{5.34}$$

定义反模糊化函数 $f(\underline{x})$ 为

$$f(\underline{x}) = \frac{\sum_{k=1}^{m} b^k \delta^k \left[\prod_{i=1}^{n} \mu_{A_i^k}(x_i)\right]^{1-\gamma-\gamma/n}}{\sum_{k=1}^{m} \delta^k \left[\prod_{i=1}^{n} \mu_{A_i^k}(x_i)\right]^{1-\gamma+\gamma/n}} \tag{5.35}$$

因此，该系统是由单值模糊产生器、高斯型隶属函数、乘积推理规则、消极-积极补偿运算以及改进型重心反模糊化器构成的补偿模糊神经网络.

(4) 补偿模糊神经网络的学习算法

对 n 维输入数据 x^p，$[x^p = (x_1^p, \cdots, x_n^p)]$ 和一维输出数据 y^p，$p=1, 2, \ldots, N$. 设计一个训练算法去最优调整模糊神经系统输入与输出隶属函数的中心和宽度.

模糊子集 A_i^k 和 B^k 的模糊隶属函数定义如下式：

$$\mu_{A_i^k}(x_i^p) = \exp\left[-\left(\frac{x_i^p - a_i^k}{\sigma_i^k}\right)^2\right] \tag{5.36}$$

$$\mu_{B^k}(y^p) = \exp\left[-\left(\frac{y^p - b^k}{\delta^k}\right)^2\right] \tag{5.37}$$

由式(5.35)得

$$f(x^p) = \frac{\sum_{k=1}^{m} b^k \delta^k z^k}{\sum_{k=1}^{m} \delta^k z^k} \tag{5.38}$$

其中 $z^k = \left[\prod_{i=1}^{n} \mu_{A_i^k}(x_i^p)\right]^{1-\gamma+\gamma/n}$. (5.39)

目标函数定义为

$$E^p = \frac{1}{2}[f(x^p) - y^p]^2 \tag{5.40}$$

根据梯度下降方法，有

1) 训练输出隶属函数的中心：

$$b^k(t+1)=b^k(t)-\eta\frac{\partial E^p}{\partial b^k}\Big|_t = b^k(t)-\eta\frac{[f(x^p)-y^p]\delta^k z^k}{\sum_{k=1}^{m}\delta^k z^k}\Big|_t \quad (5.41)$$

2)训练输出隶数函数的宽度：

$$\delta^k(t+1)=\delta^k(t)-\eta\frac{\partial E^p}{\partial \delta^k}\Big|_t = \delta^k(t)-\eta\frac{[f(x^p)-y^p][b^k-f(x^p)]z^k}{\sum_{k=1}^{m}\delta^k z^k}\Big|_t \quad (5.42)$$

3)训练输入隶数函数的中心：

$$\frac{\partial E^p}{\partial \sigma_i^k}=\frac{2[f(x^p)-y^p][b^k-f(x^p)][x_i^p-a_i^k][1-\gamma+\gamma/n]\delta^k z^k}{\sigma_i^{k^2}\sum_{k=1}^{m}\delta^k z^k}\Big|_t \quad (5.43)$$

$$a_i^k(t+1)=a_i^k(t)-\eta\frac{\partial E^p}{\partial a_i^k}\Big|_t \quad (5.44)$$

4)训练输入隶属函数的宽度：

$$\frac{\partial E^p}{\partial \sigma_i^k}=\frac{2[f(x^p)-y^p][b^k-f(x^p)][x_i^p-a_i^k]^2[1-\gamma+\gamma/n]\delta^k z^k}{\sigma_i^{k^3}\sum_{k=1}^{m}\delta^k z^k}\Big|_t \quad (5.45)$$

$$\sigma_i^k(t+1)=\sigma_i^k(t)-\eta\frac{\partial E^p}{\partial \sigma_i^k}\Big|_t \quad (5.46)$$

5)训练补偿度：

$$\gamma\in[0,1],\ 定义\ \gamma=\frac{c^2}{c^2+d^2} \quad (5.47)$$

$$\frac{\partial E^p}{\partial \gamma}=\frac{[f(x^p)-y^p][b^k-f(x^p)]\left[\frac{1}{n}-1\right]\delta^k z^k\ln\left[\prod_{i=1}^{n}\mu_{A_i^k}(x_i^p)\right]}{\sum_{k=1}^{m}\delta^k z^k}\Big|_t \quad (5.48)$$

于是

$$c(t+1)=c(t)-\eta\left\{\frac{2c(t)d^2(t)}{[c^2(t)+d^2(t)]^2}\right\}\frac{\partial E^p}{\partial \gamma}\Big|_t \quad (5.49)$$

$$d(t+1)=d(t)+\eta\left\{\frac{2d(t)c^2(t)}{[c^2(t)+d^2(t)]^2}\right\}\frac{\partial E^p}{\partial \gamma}\Big|_t \quad (5.50)$$

$$\gamma(t+1)=\frac{c^2(t+1)}{c^2(t+1)+d^2(t+1)} \quad (5.51)$$

其中 η 是学习率，$t=0,1,2,\cdots$

(5) 验证补偿模糊神经网络的性能

一个非线性系统如下式所示：

$$\dot{x}_1(t)=-x_1(t)x_2^2(t)+0.999+0.42\cos(1.75t) \quad (5.52)$$

$$x_2(t) = x_1(t)x_2^2(t) - x_2(t) \tag{5.53}$$

$$y(t) = \sin[x_1(t) + x_2(t)] \tag{5.54}$$

用MATLAB中的"ode45"函数解方程(5.52)、(5.53)和(5.54)，可以得到105个$x_1(t)$、$x_2(t)$和$y(t)$值. 其中，$t\in[0,20]$，$x_1(0)=1.0$，$x_2(0)=1.0$.

定义两输入一输出补偿神经模糊系统的输入数据为$x_1^p(t)$和$x_2^p(t)$，输出数据为$y^p(t)$，其中，$p=1,2,\cdots,105$，补偿度为γ，25个模糊IF-THEN规则描述如下：

$$FR^{(k)}: [IF\ x_1 \text{ is } A_1^k \text{ and } x_2 \text{ is } A_2^k]^{\gamma} \text{THEN y is } B^k$$

其中，$k=0,1,\cdots,24$. x_1和x_2是模糊语言变量，模糊隶属函数A_1^k，A_2^k和B^k定义如下：

$$\mu_{A_i^k}(x_i^p) = \exp\left[-\left(\frac{x_i^p - a_i^k}{\sigma_i^k}\right)^2\right] \tag{5.55}$$

$$\mu_{B^k}(y^p) = \exp\left[-\left(\frac{y^p - b^k}{\delta^k}\right)^2\right] \tag{5.56}$$

根据输入空间模糊分割方法，初始化模糊IF-THEN规则. 如图5.27所示.

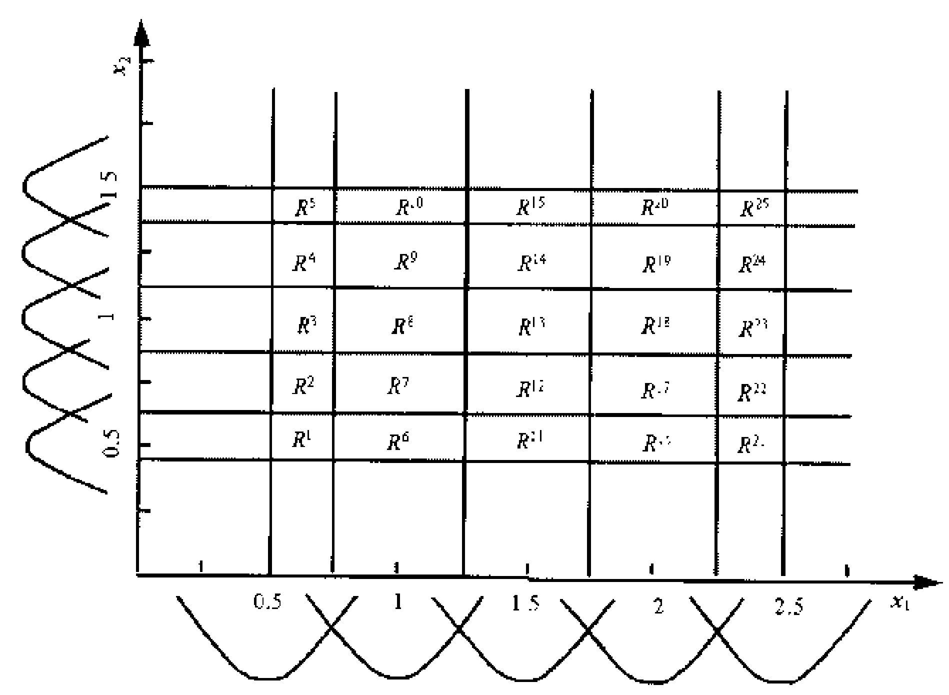


图5.27　输入空间模糊分割

当$0.5\leqslant x_1^p(t)\leqslant 2.5$，$0.4\leqslant x_2^p(t)\leqslant 1.5$，$0\leqslant y^p(t)\leqslant 1$时，模糊隶属函数$A_1^k$、$A_2^k$和$B^k$定义如下：

$$\mu_{A_1^k}(x_1) = \exp\left[-\left(\frac{x_1 - 0.5 - 0.5\lfloor k/5\rfloor}{0.5}\right)^2\right] \tag{5.57}$$

$$\mu_{A_2^k}(x_2) = \exp\left\{-\left(\frac{x_2 - 0.5 - 0.25(k\bmod 5)}{0.5}\right)^2\right\} \tag{5.58}$$

$$\mu_{B^k}(y) = \exp\left[-\left(\frac{y - b^k}{0.5}\right)^2\right] \tag{5.59}$$

其中

$$b^k = \frac{1}{N_k}\sum_{i_k=1}^{N_k} y^{M_k(i_k)}(t), \qquad N_k \neq 0 \tag{5.60}$$

$$b^k = 0.5, \qquad N_k = 0 \tag{5.61}$$

N_k 是 $M_k(i_k)$ 的总数，i_k 是一个计数器，如 $1 \leqslant i_k \leqslant N_k$，见表 5.6.

表 5.6　初始化输出隶属函数的中心

	$x_2 \in$ [0.4, 0.625]	$x_2 \in$ [0.625, 0.875]	$x_2 \in$ [0.875, 1.125]	$x_2 \in$ [1.125, 1.375]	$x_2 \in$ [1.375, 1.5]
$x_1 \in$ [0.5, 0.75]	M_0 (t_0), N_0	M_1 (t_1), N_1	M_2 (t_2), N_2	M_3 (t_3), N_3	M_4 (t_4), N_4
$x_1 \in$ [1.75, 1.25]	M_5 (t_5), N_5	M_6 (t_6), N_6	M_7 (t_7), N_7	M_8 (t_8), N_8	M_9 (t_9), N_9
$x_1 \in$ [1.25, 1.75]	M_{10} (t_{10}), N_{10}	M_{11} (t_{11}), N_{11}	M_{12} (t_{12}), N_{12}	M_{13} (t_{13}), N_{13}	M_{14} (t_{14}), N_{14}
$x_1 \in$ [1.75, 2.25]	M_{15} (t_{15}), N_{15}	M_{16} (t_{16}), N_{16}	M_{17} (t_{17}), N_{17}	M_{18} (t_{18}), N_{18}	M_{19} (t_{19}), N_{19}
$x_1 \in$ [2.25, 2.5]	M_{20} (t_{20}), N_{20}	M_{21} (t_{21}), N_{21}	M_{22} (t_{22}), N_{22}	M_{23} (t_{23}), N_{23}	M_{24} (t_{24}), N_{24}

(6) 两种神经网络性能比较

比较常规模糊神经网络与补偿模糊神经网络的性能时，主要观察网络训练步数与误差收敛的情况. 用上述介绍的补偿学习算法训练该非线性系统，两种模糊神经网络均包含 25 个模糊规则，均采用单值模糊产生器、高斯型隶属函数、乘积推理规则以及改进型重心反模糊化器. 仿真结果显示：具有补偿学习算法的补偿神经模糊网络在训练步数、训练时间及其误差精度等方面都优于常规的模糊神经网络，其学习收敛速度快，误差曲线也更加稳定. 两种网络的误差和训练步数性能如表 5.7 所示.

表 5.7　全局函数与训练步数对照表

训练步数 ＼ 误差 E	0.02	0.01	0.005
模糊神经网络	5	15	25
补偿模糊神经网络（补偿度 0.5）	2	7	12

两种网络误差-步数曲线和训练数据-网络输出曲线如图 5.28～5.31 所示.

补偿模糊神经网络的程序清单如下：

```
clear;
tspan=[0 20];x0=[1.0 1.0];
global kk;kk=0;
[t,x]=ode45('cb',tspan,x0),kk              %解方程
x1=x(:,1);x2=x(:,2);
siz=size(x);                                %求输入矩阵 x 的维数
nu=siz(:,1);                                %求输入矩阵 x 的行数,即输入模式对数
for d=1:1:nu
  y(d,1)=sin(x1(d,:)+x2(d,:));              %求函数 y 的解
end
rule=25;                                    %定义模糊规则数
```

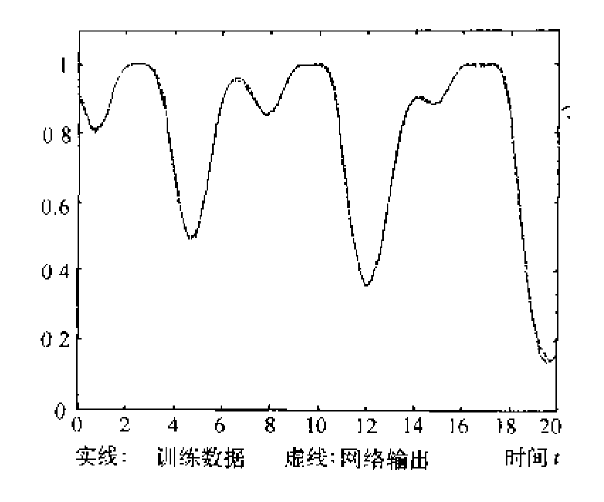


图 5.28 模糊网络训练数据-网络输出曲线



图 5.29 模糊网络训练步数-全局误差曲线

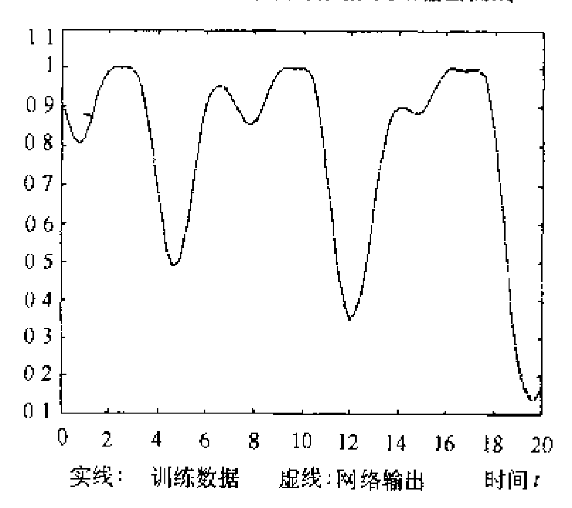


图 5.30 补偿模糊网络训练数据-网络输出曲线

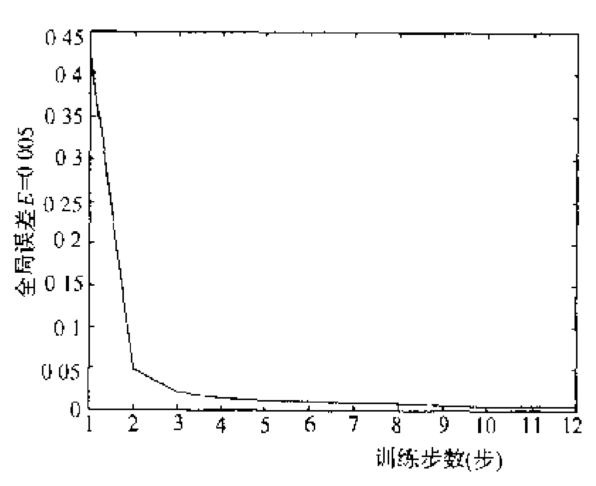


图 5.31 补偿模糊网络训练步数-全局误差曲线

```
uA=zeros(rule,2);
uA1=uA(:,1);
uA2=uA(:,2);
c=zeros(rule,2);
c1=c(:,1);c2=c(:,2);number=0;
for i=1:1:rule
    c1(i,1)=0.5;c2(i,1)=0.5;              %定义初始输入隶属函数宽度
    d(i,:)=0.5;                           %定义初始输出隶属函数宽度
  g(i,1)=0.1;
  h(i,1)=0.1;
  m(i,1)=(g(i,:)^2)/(g(i,1)^2+h(i,1)^2);          %定义初始补偿度
  q(i,:)=1-m(i,:)+m(i,:)/2;
  a1(i,1)=0.5+0.5*ceil(i/5);              %定义初始输入隶属函数中心
  a2(i,1)=0.5+0.25*rem(i,5);              %定义初始输入隶属函数中心
end
```

```
n=0.95;                                    %定义学习率
ww=zeros(rule,1);mkik=zeros(rule,1);b=zeros(rule,1);
xspan1=[0.5,0.75,1.25,1.75,2.25,2.5];      %定义输入空间模糊分割
xspan2=[0.4,0.625,0.875,1.125,1.375,1.5];  %定义输入空间模糊分割
for s=1:1:nu
  x11=x1(s,:);x22=x2(s,:);yy=y(s,:);k=0;
  for i1=1:1:5
      for i2=1:1:5
              k=k+1;
              if(x11>=xspan1(i1)&x11<=xspan1(i1+1)&
              x22>=xspan2(i2)&x22<=xspan2(i2+1))
              ww(k,1)=ww(k,1)+1;
              mkik(k,1)=yy^s;
              b(k,:)=b(k,:)+mkik(k,1);
              end
      end
  end
end
for i=1:1:rule
      if (ww(i,1)==0)
        b(i,:)=0.5;
      else
          b(i,:)=b(i,:)/ww(i,1);       %定义初始输出隶属函数中心
      end
end
e=ones(1,nu);eee=1;          %定义初始误差
while(eee>=0.005)
eeh=0;
for s=1:1:nu        %105 个输入模式对依次输入
  x11=x1(s,:);x22=x2(s,:);yy=y(s,:);
  s1=0;s2=0;
  for i=1:1:rule
  uA1(i,:)=exp(-((x11-a1(i,:))/c1(i,:))^2); %计算输入隶属函数
  uA2(i,:)=exp(-((x22-a2(i,:))/c2(i,:))^2); %计算输入隶属函数
  z(i,:)=(uA1(i,:)*uA2(i,:))^q(i,:);
  s1=s1+b(i,:)*d(i,:)*z(i,:);
  s2=s2+d(i,:)*z(i,:);
end
f(s,:)=s1/s2;
```

```
e(1,s)=((f(s,:)-yy)^2)/2;          %计算目标函数
eeh=eeh+e(1,s);                    %计算全局误差
for j=1:1:rule
  %计算输出隶属函数的中心
  dyb(j,:)= -((f(s,:)-yy)*d(j,:)*z(j,:))/s2;
  %计算输出隶属函数的宽度
  dyd(j,:)=-((f(s,:)-yy)*(b(j,:)-f(s,:))*z(j,:))/s2;
  %计算输入隶属函数的中心
  dya1(j,:)=-(2*(f(s,:)-yy)*(b(j,:)-f(s,:))*(x11-
           a1(j,:))*q(j,:)*d(j,:)*z(j,:))
           /((c1(j,:)^2)*s2);
  dya2(j,:)=-(2*(f(s,:)-yy)*(b(j,:)-f(s,:))*(x22-
           a2(j,:))*q(j,:)*d(j,:)*z(j,:))
           /((c2(j,:)^2)*s2);
  %计算输入隶属函数的宽度
  dyc1(j,:)=-(2*(f(s,:)-yy)*(b(j,:)-f(s,:))*((x11-
           a1(j,:))^2)*q(j,:)*d(j,:)*z(j,:))
           /((c1(j,:)^3)*s2);
  dyc2(j,:)=-(2*(f(s,:)-yy)*(b(j,:)-f(s,:))*((x22-
           a2(j,:))^2)*q(j,:)*d(j,:)*z(j,:))
           /((c2(j,:)^3)*s2);
  if ((uA1(j,:)*uA2(j,:))~=0)
    tt(j,:)=-((f(s,:)-yy)*(b(j,:)-f(s,:))*d(j,:)*z(j,:)
            *(log(uA1(j,:)*uA2(j,:))))/(2*s2);
    g(j,:)=g(j,:)-(2*n*g(j,:)*(h(j,:)^2)*tt(j,:))
             /((g(j,:)^2+h(j,:)^2)^2);
    h(j,:)=h(j,:)+(2*n*h(j,:)*(g(j,:)^2)*tt(j,:))
             /((g(j,:)^2+h(j,:)^2)^2);
    m(j,:)=(g(j,:)^2)/(g(j,:)^2+h(j,:)^2);
    q(j,:)=1-m(j,:)+m(j,:)/2;
  end
%调整输入、输出隶属函数的中心和宽度
  b (j,:) =b (j,:) +n*dyb (j,:);
  d (j,:) =d (j,:) +n*dyd (j,:);
  a1 (j,:) =a1 (j,:) +n*dya1 (j,:);
  a2 (j,:) =a2 (j,:) +n*dya2 (j,:);
  c1 (j,:) =c1 (j,:) +n*dyc1 (j,:);
  c2 (j,:) =c2 (j,:) +n*dyc2 (j,:);
  end
```

```
end
    eee=eeh
    number=number+1
    num (number,:) =number;
    eeee (number,:) =eee;
end
figure (1)
plot (t, y,'r', t, f,'b')        %绘制跟踪曲线
figure (2)
plot (num, eeee)        %绘制训练步数-误差曲线
```

4. 利用补偿模糊神经网络进行故障检测

利用上述补偿模糊神经网络，对基于补偿模糊神经网络的未建模系统进行了故障检测. 重点利用模糊神经网络的递归运算，进行长时间的预报，不需要参考实际输出测量，就可以在训练数据范围内提供系统的外部输入. 因此，补偿神经网络模型就可以预报系统正常运行行为. 如果故障发生，在比较系统的测量输出与预报输出的基础上，产生残差，残差将给出实际传感器测量偏差，对残差信号进行分析，运用故障决策规则就可以进行故障检测，同时也可以利用传感器输出信息来辨识故障元件.

设某控制系统的状态和输出观测方程为

$$x(k+1)=\begin{bmatrix}(1-0.05T)x_1(k)+Tu(k)x_1^3(k)\\ -x_1(k)+7\end{bmatrix} \tag{5.62}$$

$$y(k)=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}x_1(k)\\ x_2(k)\end{bmatrix} \tag{5.63}$$

其中 $x(k)=[x_1(k)\quad x_2(k)]^T, y(k)=[y_1(k)\quad y_2(k)]^T$

$T=0.01, u(k)=0.04, x(1)=[2\quad 5]^T$.

已知：$x_1(k)$，$x_2(k)$和 $u(k)$的分布区域分别为[1.5,6.5]、[0.5,5.5]和[−0.5,0.5].

利用上述补偿模糊神经网络，对此系统进行仿真，训练18步后，状态1和状态2网络跟踪轨迹，以及增加白噪声后残差曲线如图5.32、5.33、5.34和5.35所示.

当 $k=250$ 时，传感器检测故障，其状态1和状态2残差曲线如图5.36和5.37所示.

该系统的仿真程序结构与上节所介绍的补偿模糊神经网络的仿真程序类似，只是在定义输入空间模糊分割区域和初始输入、输出隶属函数的中心和宽度时有所不同. 仿真结果表明，补偿模糊神经网络具有很好的自学习性，对已学习过的样本，网络能够迅速准确地识别，对系统故障敏感.

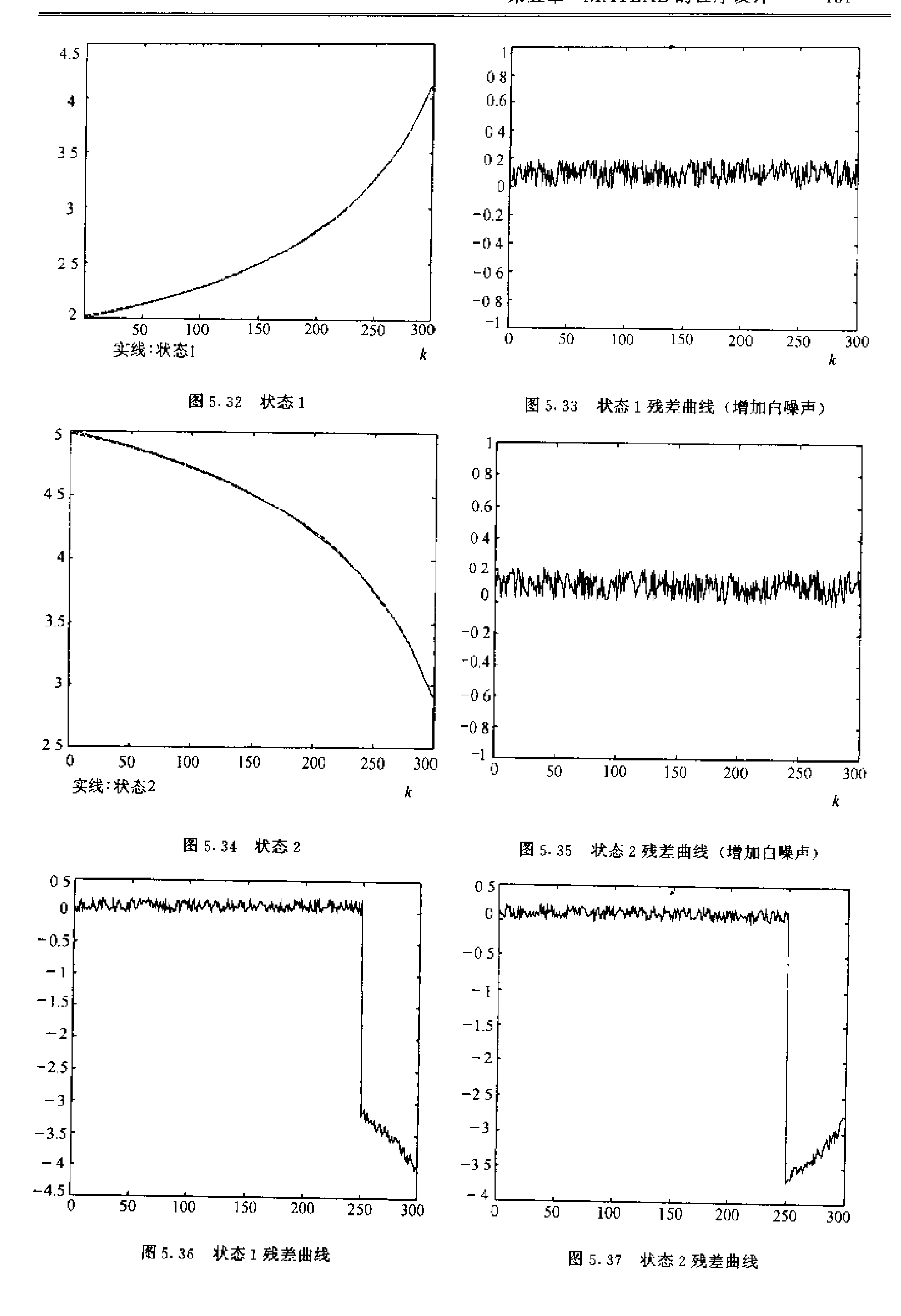


图 5.32　状态 1

图 5.33　状态 1 残差曲线（增加白噪声）

图 5.34　状态 2

图 5.35　状态 2 残差曲线（增加白噪声）

图 5.36　状态 1 残差曲线

图 5.37　状态 2 残差曲线

第六章 感 知 器

感知器（perceptron）是由美国学者罗森布拉特（F. Rosenblatt）于1957年提出的，其目的是为了模拟人脑的感知和学习能力. 感知器是最早提出的一种神经网络模型. 它是一个具有单层神经元的网络，网络的激活函数是线性阈值单元. 原始的感知器算法只有一个输出结点，它相当于单个神经元.

早期的研究人员试图用感知器模拟人脑的感知特征，但后来发现感知器的学习能力有很大的局限（如只能对线性可分的输入向量进行分类），以至于人们对它的能力和应用前景得出了十分悲观的结论. 尽管如此，这种神经网络模型的出现对早期神经网络的研究，以及后来许多神经网络的出现产生了极大的影响. 就目前来看，它仍然是一种很有用的神经网络模型. 感知器特别适用于简单的模式分类问题. 当它用于两类模式分类时，相当于在高维样本空间中，用一个超平面将两类样本分开. Rosenblatt 已证明，如果两类模式是线性可分的（指存在一个超平面将它们分开），则算法一定是收敛的.

另外 Rosenblatt 除了研究单层感知器外，还研究了有一个隐层的感知器. 但当时他只能证明单层感知器可以将线性可分的输入向量进行正确分类. 本书所提到的感知器都是指单层感知器. 本章所讨论的内容是在 MATLAB5.3 版本基础上进行的。

6.1 感知器的原理

6.1.1 感知器神经元模型

感知器神经元模型如图 6.1 所示，其中 R 是网络的输入个数.

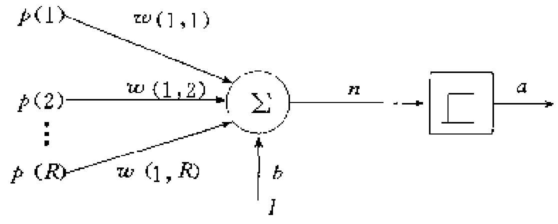


图 6.1 感知器神经元模型

感知器神经元的每一个输入 $p(i)(i=1,\cdots,R)$都对应一个相应的权值 $w(1,i)$，所有的输入与其对应的权值的乘积之和输入给一个线性阈值单元. 该线性阈值单元的另一个输入是常数 1 乘以阈值 b. 由于感知器的激活函数是符号函数阈值单元，因此感知器可以将输入向量分为两个区域. 最简单的一种情形是当输入大于等于 0 时输出为 1，当输入小于 0 时输出为 0. MATLAB5.3 提供的一种线性阈值单元为 hardlim，如图 6.2 所示.

如果以两个输入的单层感知器神经元为例，对于选定的权值 $w(1,1)=-1, w(1,2)$

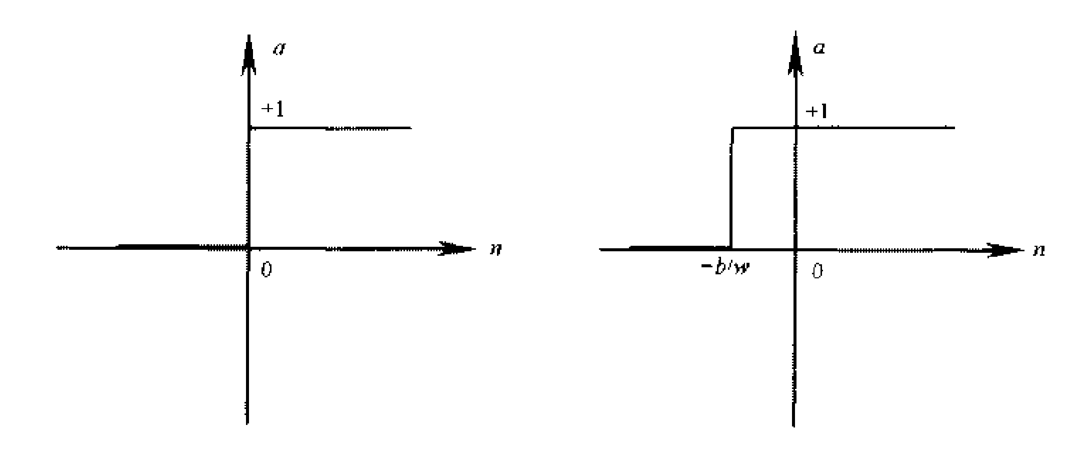


图 6.2 线性阈值单元 hardlim( )

=2 和阈值 $b=1$，分类情况如图 6.3 所示.

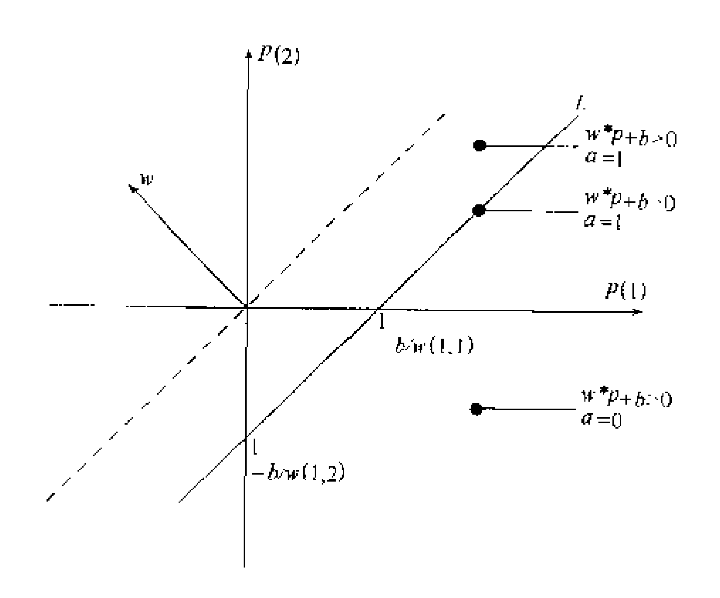


图 6.3 输入矢量的分类情况

从图中可见，分割线 L 将两个输入矢量所组成的平面分为两个区域：直线上部的输入矢量使感知器的输出为 1，直线下部的输入矢量使感知器的输出为 0.

6.1.2 感知器神经元网络的网络结构

图 6.4 描述了感知器神经元网络结构的两种形式. 感知器神经元网络是由 hardlim 产生的符号函数阈值元件组成，该网络是由单层的 S 个神经元构成，网络有 R 个输入，通过一组权值 $w(1,i)$ 与 S 个神经元相连，网络的输出为 S 个.

从图 6.4 可以看出，感知器神经网络只有一层神经元. 这是因为感知器的学习规则只能训练单层网络，根据网络结构，可以写出第 i 个输出神经元（$i=1$，…，S）的输出：

$$n(i) = \sum_{j=1}^{R} w(i,j) \times p(j) + b(i)$$

$$a(i) = f[n(i)]$$

感知器中使用了符号阈值函数单元，因此使感知器能够将输入向量分为两个区域，则有

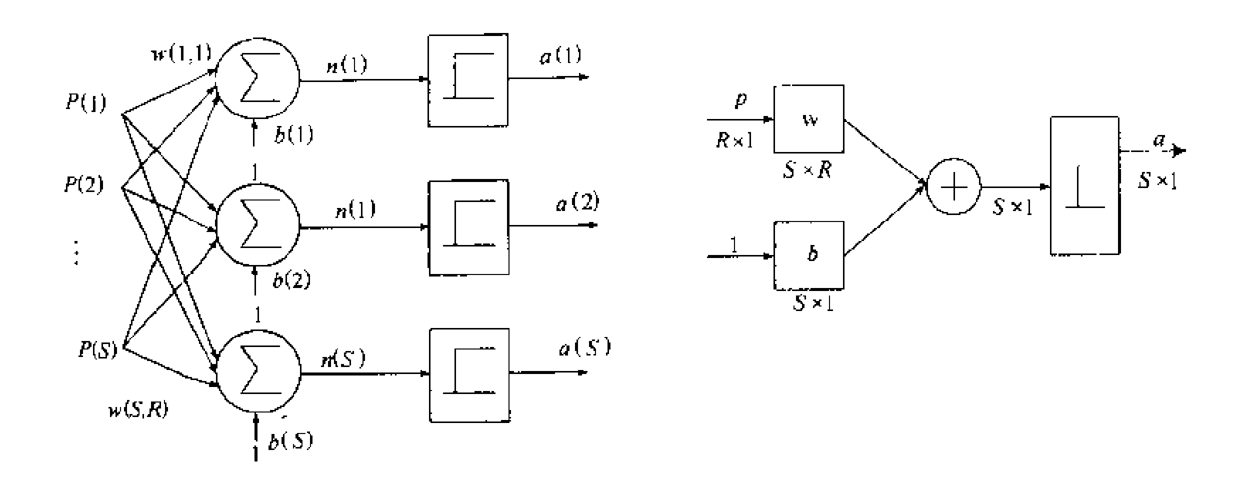


图 6.4　感知器神经元结构

$$a(i)=\begin{cases}1, & n(i)\geq 0\\ 0, & n(i)<0\end{cases}$$

当输入 $n(i)\geq 0$ 时,感知器输出 $a(i)$ 为 1,否则输出 $a(i)$ 为 0.

6.1.3　感知器神经网络的初始化

进行程序设计的第一步就是初始化,对感知器神经网络的初始化可采用 MATLAB5.3 神经网络工具箱中的 init(　)函数. 使用 init(net)函数可以得到一个已初始化的神经网络 net,该网络的权值和阈值是按照网络初始化函数来进行修整的,而网络的初始化函数是由 NET.initFcn 设定,其参数是由 NET.initParam 指定. 初始化函数的用法为

$$\text{net} = \text{init(net)}$$

MATLAB5.3 工具函数中的 newp 函数可以生成一个感知器网络,其中使用了上述语句对网络 net 进行初始化.

例 6.1　用 init(net)函数设计一个具有 2 个输入的单个神经元感知器. 输入范围是:[0,1]和[−2,2]. 设计好感知器网络之后,要求显示其权值和阈值.

```
net = newp([0 1;-2 2],1);
net.iw{1,1}
net.b{1}
```

上述语句执行之后,网络的权值和阈值被设置为 0.

对于一个已经训练过的网络,网络的权值和阈值发生了变化,可以用 init 函数重新初始化该网络的权值和阈值:

```
net = init(net);
net.iw{1,1}
net.b{1}
```

使用上述语句之后,网络的权值和阈值重新设置为 0,这是感知器网络所使用的初始值.

6.1.4 感知器神经网络的学习规则

学习规则是用来计算阈网络的新的权值和阈值的算法，感知器利用其学习规则来调整网络的权值和阈值，使该网络的输出最终达到目标的期望值.

对于输入向量 p，输出向量 a，目标矢量为 t 的感知器，该感知器的学习误差为 e，则 $e=t-a$. 此时感知器的权值阈值修正公式为

$$\Delta w(i,j) = [t(i) - a(i)] * p(j) = e(i) * p(j)$$

$$\Delta b(i) = [t(i) - a(i)] * 1 = e(i) * 1$$

式中 $i=1, 2, \cdots, S$，$j=1, 2, \cdots, R$，则更新的权值与阈值为

$$w(i,j) = w(i,j) + \Delta w(i,j)$$

$$b(i) = b(i) + \Delta b(i)$$

用矩阵来表示相应的权值与阈值更新公式：

$$w=w+e*p^{\mathrm{T}}$$

$$b=b+e$$

感知器的学习规则属于梯度下降法，该法则已被证明：如果能存在，则算法在有限次的循环迭代后可以收敛到正确的目标矢量.

在 MATLAB5.3 的神经网络工具箱中，感知器的学习规则可用函数 learnp.m 及 learnpn.m 实现.

6.1.5 感知器神经网络的训练

神经网络在应用之前必须经过训练，由此决定网络的权值和阈值. 感知器的训练过程为

对于给定的输入向量 p，计算网络的实际输出 a，并与相应的目标向量 t 进行比较，用得到的误差 e，根据学习规则进行权值和阈值的调整；重新计算网络在新权值作用下的输入，重复权值调整过程，直到网络的输出与目标值相等或者训练的次数达到预先设定的最大值时结束训练.

经过训练的网络需要验证其是否能够达到要求（即网络是否训练成功）. 若网络训练成功，那么训练后的网络对于被训练的每一组输入向量都能产生一组对应的期望输出；若在给定的最大训练次数内，网络未能完成在给定输入向量 p 的作用下，满足 $a=t$ 的要求，那么可以增加训练的次数，继续训练网络；若在足够多次的训练后，网络仍然达不到要求，那么需要分析一下，感知器神经网络是否适合解决目前这个问题. 感知器并非适用于任何分类问题. 下一节我们将介绍感知器的局限性.

在 MATLAB5.3 的神经网络工具箱中，感知器的训练函数为 adaptwb.m 和 trainwb.m.

6.1.6 感知器的局限性

由于感知器神经网络在结构和学习规则上的局限性，其应用被限制在一定的范围内. 一般来说感知器有以下局限性：

1）由于感知器的激活函数是阈值函数，则感知器神经网络的输出只能取 0 或 1. 因此感知器只能用于简单的分类问题；

2）感知器神经网络只能对线性可分的向量集合进行分类. 理论上已经证明，只要输入向量是线性可分的，感知器在有限的时间内总能达到目标向量. 但是如何确定输入向量是否线性可分，至今还没有有效的解决方法，尤其当输入向量增多时，更难以确定. 一般只有设置一定的循环次数，对网络进行训练而判定它是否能被线性可分；

3）当感知器神经网络的所有输入样本中存在奇异的样本，即该样本向量同其他所有的样本向量比较起来特别大或特别小时，网络训练所花费的时间将很长. 例如当输入和目标向量分别为

$$p=[-0.5\ -0.5\ 0.3\ -0.1\ -80;$$
$$-0.5\ 0.5\ -0.5\ 1.0\ 100];$$
$$t=[1\ 1\ 0\ 0\ 1];$$

由于输入的第五组数远远大于其他输入数据，这必然导致训练的困难. 解决此问题的方法是采用标准化感知器学习规则. 在神经网络工具箱中，实现标准化感知器学习规则的函数是 learnpn.m，其具体原理将在下面介绍.

6.1.7　多层感知器

为了解决单层感知器的局限性，60 年代末人们曾致力于该问题的研究，并找到了解决的方法——采用多层网络结构. 这样，对于单层感知器解决不了的“异或”问题，可以用两层网络结构得以解决. 如图 6.5 为一种常用的双层感知器神经网络. 其第一层是随机感知层，第二层为学习感知层.

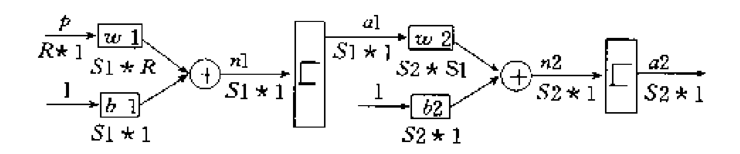


图 6.5　双层感知器网络结构图

6.2　有关感知器的神经网络工具函数

6.2.1　MATLAB 中有关感知器的工具函数

MATLAB5.3 提供了许多进行神经网络设计和分析的工具函数，这给用户带来了极大的方便. 有关这些工具函数的使用可以通过 Help 命令得到. 下表给出 MATLAB5.3 中与感知器相关的神经网络重要工具函数.

表 6.1　感知器网络的重要工具函数

函数名称	功　能
newp	生成一个感知器神经网络
hardlim	硬限幅传递函数

续表

函 数 名 称	功 能
hardlims	对称硬限幅传递函数
dotprod	权值点积函数
netsum	网络输入求和函数
initlay	某层的初始化函数
initwb	某层的权值和阈值的初始化函数
initzero	零权值阈值初始化函数
init	一个网络的初始化函数
mae	求平均绝对误差性能函数
learnp	感知器的学习函数
learnpn	标准感知器的学习函数
adaptwb	网络的权值阈值的自适应函数
adapt	神经网络的自适应
trainwb	网络的权值和阈值训练函数
train	神经网络训练函数
sim	神经网络仿真函数

6.2.2 工具函数详解

1. 神经网络函数

(1) newp 生成一个感知器神经网络

格式：net = newp (pr, s, tf, lf)

说明：感知器是用于解决简单的（即线性可分的）分类问题．newp 的输入参数为

pr　　R 个输入向量的最大值和最小值的取值范围．

s　　神经元的个数．

tf　　网络的传递函数，缺省值为 hardlim．

lf　　网络的学习函数，缺省值为 learnp．

该函数执行之后返回一个新的感知器网络．传递函数 tf 可以选取 hardlim 或 hardlims 函数．学习函数 lf 可以选取 learnp 或 learnpn 函数．

可以调用不带参数的 newp 函数在一个对话窗中定义网络的属性．

例 6.2　设计一个 2 输入的单个神经元的感知器网络，2 个输入向量的范围分别是[0 1]及[−2 2]．(这里仅仅给出了 newp 的 2 个参数，感知器使用缺省的学习函数 learnp 及传递函数 hardlim．)

net = newp ([0 1; −2 2], 1);

例 6.3　对上述网络用一组输入向量 P 和一个目标 T：

P= { [0; 0] [0; 1] [1; 0] [1; 1]};

T= {0 0 0 1};

使 P 和 T 实现与门的功能. 定义网络的训练次数为 10 次.

```
net = newp ( [0 1; -2 2], 1);
P = { [0; 0] [0; 1] [1; 0] [1; 1]};
T = {0 0 0 1};
net.adaptParam.passes = 10;
net = adapt (net, P, T);
Y = sim (net, P)
```

该程序的仿真结果为

```
Y = [0] [0] [0] [1]
```

例 6.4 对于上述的网络重新定义 P 和 T，以实现与门的功能. 其相应的输入 P 与目标 T 为

```
P = [0 0 1 1; 0 1 0 1];
T = [0 1 1 1];
```

首先对感知器进行初始化，使网络的权值和阈值均为零；仿真此时网络的输出；设置训练的最大次数为 20 次；之后重新仿真该网络：

```
net = init (net);
Y = sim (net, P)
net.trainParam.epochs = 20;
net = train (net, P, T);
Y = sim (net, P)
```

该程序的仿真结果为

```
Y = [0] [1] [1] [1]
```

注意：在一个有限的时间范围内，感知器可以对线性可分的输入向量进行分类. 如果输入向量具有奇异值，标准感知器学习函数 learnpn 比感知器学习函数 learnp 学习速度快.

感知器的特性：感知器是一个包含 dotprod 点积函数、netsum 网络输入函数及特定的传递函数的单层网络. 该层具有输入的权值和阈值. 权值和阈值的初始化函数是 initzero. 可用 adaptwb 或 trainwb 函数来适应或训练网络. 这两个函数都可以用特定的学习函数修正网络的权值和阈值. 性能参数是用 mae 函数计算的.

(2) sim 网络仿真

格式：
```
[Y, Pf, Af] = sim (net, P, Pi, Ai)
[Y, Pf, Af] = sim (net, {Q TS}, Pi, Ai)
[Y, Pf, Af] = sim (net, Q, Pi, Ai)
```

说明：sim 可仿真一个神经网络. sim 的参数为

NET　　神经网络.

P　　网络的输入.

Pi　　初始输入延迟，缺省值为 0.

Ai　　初始的层延迟，缺省值为 0.

且该函数返回：

Y　　网络的输出.

Pf　　最终输出延迟.

Af　　最终的层延迟.

注意这些参数Pi，Ai，Pf 及 Af 是可任选的，并且对于具有输入延迟或层的延迟的网络是必须使用的.

sim 的参数可有两种格式：向量单元和矩阵.

1）向量单元的格式

对于多输入多输出的网络输入序列描述为

P　　Ni×TS 元数组（cell array），每个元素 P {i，ts} 是一个 Rix×Q 的矩阵.

Pi　　Ni×ID 元数组，每个元素 Pi {i，k} 是一个 Ri×Q 的矩阵.

Ai　　Nl×LD 元数组，每个元素 Ai {i，k} 是一个 Si×Q 的矩阵.

Y　　NO×TS 元数组，每个元素 Y {i，ts} 是一个 Ui×Q 的矩阵.

Pf　　Ni×ID 元数组，每个元素 Pf {i，k} 是一个 Ri×Q 的矩阵.

Af　　Nl×LD 元数组，每个元素 Af {i，k} 是一个 Si×Q 的矩阵.

其中

Ni = net.numInputs

Nl = net.numLayers，

No = net.numOutputs

ID = net.numInputDelays

LD = net.numLayerDelays

TS = 训练的最大迭代次数

Q = 输入数据的个数

Ri = net.inputs {i} .size

Si = net.layers {i} .size

Ui = net.outputs {i} .size

Pi，Pf，Ai 及 Af 的阵列是用最初的延迟对于当前的延迟确定的：

Pi {i，k} = 在 ts=k−ID 时刻的第 i 个输入.

Pf {i，k} = 在 ts=TS+k−ID 时刻的第 i 个输入.

Ai {i，k} = 在 ts=k−LD 时刻的第 i 层输出.

Af {i，k} = 在 ts=TS+k−LD 时刻的第 i 层输出.

2）矩阵格式

每个矩阵的参数是由单个矩阵中储存的相应元数组变量的元素构成的：

P　　（Ri 的总和）×Q 的矩阵.

Pi　　（Ri 的总和）×（ID×Q）的矩阵.

Ai　　（Si 的总和）×（LD×Q）的矩阵.

Y　　（Ui 的总和）×Q 的矩阵.

Pf　　（Ri 的总和）×（ID×Q）的矩阵.

Af　　（Si 的总和）×（LD×Q）的矩阵.

[Y，Pf，Af] = sim（net，{Q TS}，Pi，Ai）用在没有输入的网络，使用元数组的

格式，如 Hopfield 网络.

[Y, Pf, Af] = sim (net, Q, Pi, Ai) 用在没有输入的网络，当使用矩阵的格式，如 Hopfield 网络.

例 6.5 设计一个有 2 个输入的单层感知器（输入范围是 [0 1]），并对一个单独的向量、3 组向量及 2 对向量：

```
p1 = [0.2; 0.9];
p2 = [ 0.2 0.5 0.1; 0.9 0.3 0.7];
p3 = { [0.2; 0.9] [0.5; 0.3] [0.1; 0.7]};
```

仿真.

```
net = newp ( [0 1; 0 1], 1);
p1 = [0.2; 0.9]; a1 = sim (net, p1)
p2 = [0.2 0.5 0.1; 0.9 0.3 0.7]; a2 = sim (net, p2)
p3 = { [0.2; 0.9] [0.5; 0.3] [0.1; 0.7]}; a3 = sim (net, p3)
```

例 6.6 用 newlind 建立含有 3 个输入 2 个神经元的线性层. 输入向量为

p1 = { [2; 0.5; 1] [1; 1.2; 0.1]}.

构造 2 个输入向量的线性层，该输入向量用默认的初始输入延迟条件（均为 0).

```
net = newlin ( [0 2; 0 2; 0 2], 2, [0 1]);
p1 = { [2; 0.5; 1] [1; 1.2; 0.1]};
 [y1, pf] = sim (net, p1)
```

算法：sim 使用以下工具来仿真一个网络 net：NET. numInputs, NET. numLayers, NET. outputConnect, NET. biasConnect, NET. inputConnect, NET. layerConnect

这些工具决定了网络的权值和阈值及与权值相关的延迟数值：

```
NET.inputWeights{i,j}.value
NET.layerWeights{i,j}.value
NET.layers{i}.value
NET.inputWeights{i,j}.delays
NET.layerWeights{i,j}.delays
```

这些函数工具指出如何用 sim 对输入得到第 i 层的输出.

```
NET.inputWeights{i,j}.weightFcn
NET.layerWeights{i,j}.weightFcn
NET.layers{i}.netInputFcn
NET.layers{i}.transferFcn
```

(3) init 初始化神经网络

格式：net = init (net)

说明：使用 init (net) 函数可以得到一个神经网络 net，该网络的权值和阈值是按照网络初始化函数来进行修整的. 而网络的初始化函数是由 NET. initFcn 设定，其参数是由 NET. initParam 确定.

例 6.7 设计一个具有 2 个输入的单个神经元感知器. 输入范围是：[0, 1] 和 [−2, 2]. 设计好感知器网络之后，要求显示其权值和阈值. 对上述网络按照输入矢量

矩阵 P 及目标矢量 T 训练网络：

```
P = [0 1 0 1; 0 0 1 1];
T = [0 0 0 1];
```

修整网络的权值和阈值，再用 init 函数重新初始化网络的权值和阈值.

```
net = newp([0 1;-2 2],1);
net.iw{1,1}
net.b{1}
P = [0 1 0 1; 0 0 1 1];
T = [0 0 0 1];
net = train(net,P,T);
net.iw{1,1}
net.b{1}
net = init(net);
net.iw{1,1}
net.b{1}
```

运行上述程序，网络的权值和阈值最终被设置为 0，这是感知器网络所使用的初始值（参见 newp 函数）.

算法：init 函数调用 NET. initFcn 函数，根据 NET. initParam 设定的参数对网络的权值和阈值进行初始化. 通常 NET. initFcn 设置为 initlay 函数，该函数按照 NET. layers {i}. initFcn 来初始化其每一层的权值和阈值.

BP 网络的 NET. layers {i}. initFcn 设置为 initnw 函数，该函数用 Nguyen-Widrow 初始化方法设计第 i 层的权值和阈值.

其他类型的网络中 NET. layers {i}. initFcn 设为 initwb 函数，其初始化一个权值和阈值是采用特定的初始化函数. 通常权值和阈值的初始化函数是 rands 函数，该函数可以产生-1 到 1 之间的随机数.

(4) adapt 神经网络的自适应

格式：[net, Y, E, Pf, Af] = adapt (NET, P, T, Pi, Ai)

说明：函数 adapt 的各个参数的含义为

NET　　一个神经网络.

P　　网络的输入.

Pi　　初始输入延迟，缺省值为 0.

Ai　　初始的层延迟，缺省值为 0.

且返回一个具有适应功能 NET. adaptFcn 及适应参数 NET. adaptParam 的结果：

NET　　修正后的网络.

Y　　网络的输出.

E　　网络的误差.

Pf　　最终输出延迟.

Af　　最终的层延迟.

注意参数 T 仅对需要目标的网络是必须的，而且是可任选的. Pi 及 Pf 仅用于具有输

人或层间的延迟的网络，而且也是可任选的.

adapt 的参数可有两种格式：向量单元和矩阵.

1）向量的格式 对于多输入多输出的网络输入序列描述为

P　　Ni×TS 元数组，每个元素 P {i，ts} 是一个 Ri×Q 的矩阵.

T　　Nt×TS 元数组，每个元素 P {i，ts} 是一个 Vi×Q 的矩阵.

Pi　　Ni×ID 元数组，每个元素 Pi {i，k} 是一个 Ri×Q 的矩阵.

Ai　　Nl×LD 元数组，每个元素 Ai {i，k} 是一个 Si×Q 的矩阵.

Y　　No×TS 元数组，每个元素 Y {i，ts} 是一个 Ui×Q 的矩阵.

Pf　　Ni×ID 元数组，每个元素 Pf {i，k} 是一个 Ri×Q 的矩阵.

Af　　Nl×LD 元数组，每个元素 Af {i，k} 是一个 Si×Q 的矩阵.

其中

Ni = net.numInputs

Nl = net.numLayers

No = net.numOutputs

Nt = net.numTargets

ID = net.numInputDelays

LD = net.numLayerDelays

TS = 训练的最大迭代次数

Q = 输入数据的个数

Ri = net.inputs{i}.size

Si = net.layers{i}.size

Ui = net.outputs{i}.size

Vi = net.targets{i}.size

Pi，Pf，Ai 及 Af 的阵列是用最初的延迟条件对于当前的延迟条件确定的：

Pi {i，k} = 在 ts=k−ID 时刻的第 i 个输入.

Pf {i，k} = 在 ts=TS+k−ID 时刻的第 i 个输入.

Ai {i，k} = 在 ts=k−LD 时刻的第 i 层输出.

Af {i，k} = 在 ts=TS+k−LD 时刻的第 i 层输出.

2）矩阵格式

每个矩阵的参数是由单个矩阵中储存的相应元数组变量的元素构成的：

P　　（Ri 的总和）×Q 的矩阵.

T　　（Vi 的总和）×Q 的矩阵.

Pi　　（Ri 的总和）×（ID×Q）的矩阵.

Ai　　（Si 的总和）×（LD×Q）的矩阵.

Y　　（Ui 的总和）×Q 的矩阵.

Pf　　（Ri 的总和）×（ID×Q）的矩阵.

Af　　（Si 的总和）×（LD×Q）的矩阵.

例 6.8 有 12 组两个输入序列（其中 t1 是与 p1 相对应的）用于确定一个滤波器运行.

```
p1 = {-1 0 1 0 1 1 -1 0 -1 1 0 1};
t1 = {-1 -1 1 1 1 2 0 -1 -1 0 1 1};
```

newlin 用于建立单个神经元的一个输入层(输入范围是[-1 1]),延迟输入为0或1,且学习率为0.5. 对这个线性层进行仿真.

```
p1 = {-1 0 1 0 1 1 -1 0 -1 1 0 1};
t1 = {-1 -1 1 1 1 2 0 -1 -1 0 1 1};
net = newlin ( [-1 1], 1, [0 1], 0.5);
[net, y, e, pf] = adapt (net, p1, t1);
perf = mse (e)
```

此处网络将输入序列从头到尾调整一次. 网络的均方误差可显示出.(因为这是第一次调用 adapt 函数, Pi 用缺省值.)得到的仿真结果为: perf = 0.3347. 这个误差是相当大的.

用另外12个数据调整网络(用上一步的 Pf 作为新的初始延迟):

```
p2 = {1 -1 -1 1 1 -1 0 0 0 1 -1 -1};
t2 = {2 0 -2 0 2 0 -1 0 0 1 0 -1};
[net, y, e, pf] = adapt (net, p2, t2, pf);
perf = mse (e)
```

得到的仿真结果为: perf = 0.0828.

再用所有数据调整网络100次.

```
p3 = [p1 p2];
t3 = [t1 t2];
net. adaptParam. passes = 100;
[net, y, e] = adapt (net, p3, t3);
perf = mse (e)
```

得到的仿真结果为: perf = 0.1039. 通过这些数据调整网络100次之后误差就非常小了,即网络调整到输入与目标的相对应关系上.

算法: adapt 调用了由 NET. adaptFcn 指出的函数,使用的调整参数由 NET. adaptParam 指出.

给出一个包含TS步的输入序列,网络的更新如下. 输入序列的每一步一个地供给网络. 每步之后,在给出下一步的输入之前,网络的权值和阈值都被修整了,因此网络更新TS次.

(5) train 神经网络的训练函数

格式: [net, tr] = train (NET, P, T, Pi, Ai)

[net, tr] = train (NET, P, T, Pi, Ai, VV, TV)

说明: train 函数是按照 NET. trainFcn 和 NET. trainParam 训练网络 NET 的. train (NET, P, T, Pi, Ai) 中各变量的含义为

NET 神经网络.

P 网络的输入.

T 网络的目标,默认值为0.

Pi　初始输入延迟，默认值为 0.

Ai　初始的层延迟，默认值为 0.

且返回：

NET　修正后的网络.

TR　训练的记录（训练步数和性能 epoch and perf).

注意参数 T 仅对需要目标的网络是必须的，而且是可任选的. Pi 及 Ai 仅用于具有输入或层间的延迟的网络，而且也是可任选的.

train 的参数可有两种格式：向量单元和矩阵.

1）向量的格式

对于多输入多输出的网络输入序列描述为

P　Ni×TS 元数组 cell array，每个元素 P {i，ts} 是一个 Ri×Q 的矩阵.

T　Nt×TS 元数组，每个元素 P {i，ts} 是一个 Vi×Q 的矩阵.

Pi　Ni×ID 元数组，每个元素 Pi {i，k} 是一个 Ri×Q 的矩阵.

Ai　Nl×LD 元数组，每个元素 Ai {i，k} 是一个 Si×Q 的矩阵.

其中：

Ni = net.numInputs

Nl = net.numLayers

Nt = net.numTargets

ID = net.numInputDelays

LD = net.numLayerDelays

TS = 训练的最大迭代次数

Q = 输入数据的个数

Ri = net.inputs{i}.size

Si = net.layers{i}.size

Vi = net.targets{i}.size

Pi，Pf，Ai 及 Af 的阵列是用最初的延迟条件对于最近的延迟确定的：

Pi{i,k} = 在 ts=k-ID 时刻的第 i 个输入.

Pf{i,k} = 在 ts=TS+k−ID 时刻的第 i 个输入.

Ai{i,k} = 在 ts=k−LD 时刻的第 i 层输出.

Af{i,k} = 在 ts=TS+k−LD 时刻的第 i 层输出.

2)矩阵格式

每个矩阵的参数是由单个矩阵中储存的相应元数组变量的元素构成的：

P　(Ri 的总和)×Q 的矩阵.

T　(Vi 的总和)×Q 的矩阵.

Pi　(Ri 的总和)×(ID * Q) 的矩阵.

Ai　(Si 的总和)×(LD * Q) 的矩阵.

train (NET,P,T,Pi,Ai,VV,TV) 可用给定/测试向量随意构造.

VV.P，TV.P　给定/测试输入.

VV.T，VV.T　给定/测试目标，缺省值为 0.

VV.Pi，VV.Pi　　给定/测试的初始输入延迟，缺省值为0.

算法：train调用了NET.trainFcn函数，使用的参数由NET.trainParam定义.

网络的训练一直持续到满足最大的步长、达到训练的目标或者是其他由NET.trainFcn给出的终止条件.

有些训练函数不符合上述标准，而是根据给出每一步只有一个输入向量(或序列)而定.每一步训练是从输入向量(或序列)中随机地选取一个输入向量(或序列).newc与newsom建立的网络是用trainwb1函数进行网络训练的.

2. 权值函数

dotprod 权值点积函数

格式：
```
Z = dotprod(W,P)
df = dotprod('deriv')
```

说明：dotprod是一个点积权值的函数，权值与输入的点积可得到加权输入.dotprod(W,P)的输入参数为

W　　S×R维的权值矩阵，

P　　Q组R维输入向量，

且返回Q组S维的W与P的点积.

例6.9　随机地定义一个权值矩阵和输入向量，计算其相应的加权输入Z.

```
W = rand(4,3);
P = rand(3,1);
Z = dotprod(W,P)
```

仿真结果为：

```
Z =
    1.6786
    0.8540
    1.0048
    0.6012
```

网络应用：可以调用newp或newlin建立一个标准的网络，该网络使用了dotprod点积函数.对于一个网络，输入权值用dotprod进行点积，这是按照将NET.inputWeight{i，j}.weightFcn设置为dotprod实现的，而对于每一层权值的点积是将NET.inputWeight{i，j}.weightFcn设置为dotprod来实现的.

在上述任意一种情况下，可以调用sim来仿真具有传递函数为hardlim的网络，可参见newp及newlin的仿真例子.

3. 网络的输入函数

netsum 神经网络的输入求和函数

格式：
```
N = netsum (Z1, Z2,...)
df = netsum ('deriv')
```

说明：netsum是一个神经网络的输入求和函数.该函数的功能是将某一层的加权输

入和阈值相加作为该层网络的输入. 其中 netsum (Z1, Z2, ...) 可以取任意数目的输入, Zi 是 Q 组 S 维的矩阵, 并且返回对 Zi 按元素求和的结果 N . netsum ('deriv') 函数返回 netsum 的导数函数.

例 6. 10. 下面的 netsum 组合了两个加权输入向量的集合 (这是我们自己定义的), 并用 netsum 对具有阈值向量的上述两个加权输入求和.

```
z1 = [1 2 4; 3 4 1];
z2 = [-1 2 2; -5 -6 1];
n = netsum (z1, z2)
```

下面的 netsum 是对具有阈值向量的上述两个加权输入求和. 由于 z1 和 z2 分别包含 3 个向量, 所以必须用 concur 产生 b 的 3 个相同的拷贝, 这样维数才会匹配.

```
b = [0; -1];
n = netsum (z1, z2, concur (b, 3))
```

网络的应用: 通过使用 newp 或 newlin 函数, 并调用 netsum 函数可以生成一个标准网络, 并且可调用 sim 来仿真带有 netsum 的网络.

可以用 netsum 函数来改变一个网络的第 i 层. 方法是将 NET. layers {i}. netInputFcn 设置为 netsum , 之后可调用 sim 来仿真带有 netsum 的网络.

可参见 newp 或 newlin 的仿真实例.

4. 传递函数

(1) hardlim 硬限幅传输函数

格式: a = hardlim (N)

　　info = hardlim (code)

说明: hardlim 是一个传输函数. 它通过计算网络的输入得到该层的输出. 如果网络输入达到门限, 硬限幅传输函数的输出为 1, 否则输出为 0. 这表明神经网络可用于判断和分类.

hardlim (N) 的输入为

N　　Q 组 S 维的网络输入向量, 并返回该层的输出矩阵向量. 当 N 大于 0, 返回的元素是 1, 当 N 小于 0, 返回的元素是 0.

hardlim (code) 与 hardlim (N) 不同的是根据不同代码 code 返回有用的信息:

'deriv'　　导数函数名.

'name'　　传递函数的全称 .

'output'　　传递函数的输出范围.

'active'　　传递函数的输入范围.

例 6. 11　建立一个 hardlim 传递函数并且打印出该图形.

```
n = -5: 0.1: 5;
a = hardlim (n);
plot (n, a)
```

网络应用: 我们可以调用 newp 函数建立一个标准的网络, 该函数所使用缺省的传递

函数是 hardlim 函数. 为了使网络的某层使用hardlim 传递函数，可将NET. layers {i}. transferFcn 设置为hardlim .

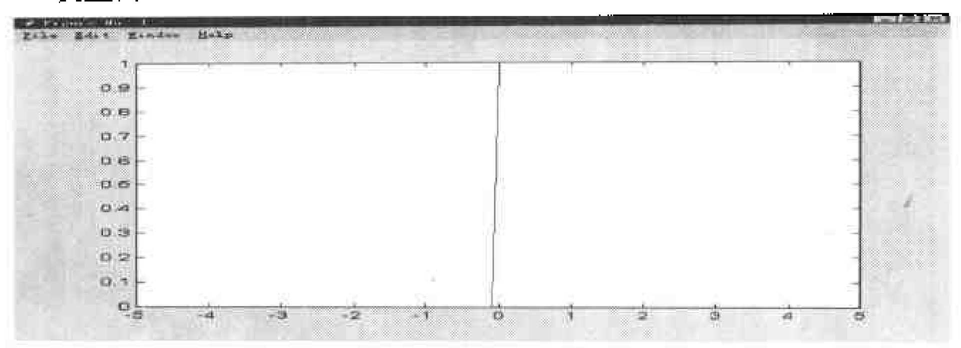


图 6.6 范例6.11的运行结果

在上述任意一种情况下，都可以调用sim 来仿真具有 hardlim 传递函数的网络. 参见 newp 函数的仿真实例.

算法：if n >= 0，则 hardlim （n） = 1，

否则 hardlim （n） = 0.

（2） hardlims 对称的硬限幅传输函数

格式：a = hardlims （N）

info = hardlims （code）

说明：hardlims 是一个传输函数，它通过计算网络的输入得到该层的输出. 如果网络输入达到门限，对称的硬限幅传输函数的输出为 1，否则为－1. 同一般的硬限幅传输函数一样，它可以用于判断和分类.

hardlims （N） 的输入为：N － Q 组 S 维的网络输入向量矩阵，返回该层的输出矩阵向量. 当 N 大于 0，返回的元素是 1，当 N 小于 0，返回的元素是－1.

hardlims （code） 与 hardlim （N） 不同的是根据不同代码 code 返回有用的信息：

'deriv' 导出函数名.

'name' 传递函数的全称 .

'output' 传递函数的输出范围.

'active' 传递函数的输入范围.

例 6.12 建立一个 hardlims 传递函数并且打印出该曲线.

```
n = -5: 0.1: 5;
a = hardlims (n);
plot (n, a)
```

网络应用：我们可以调用newp 函数建立一个标准的网络，该函数所使用的默认的传递函数是 hardlim 函数. 为了使网络的某层使用 hardlims 传递函数，可将 NET. layers{i}. transferFcn 设置为hardlims .

在上述任意一种情况下，可以调用sim 来仿真具有hardlim 传递函数的网络，参见 newp 的仿真例子.

算法：if n >= 0，则 hardlims （n） = 1，

否则 hardlims （n） = 0.

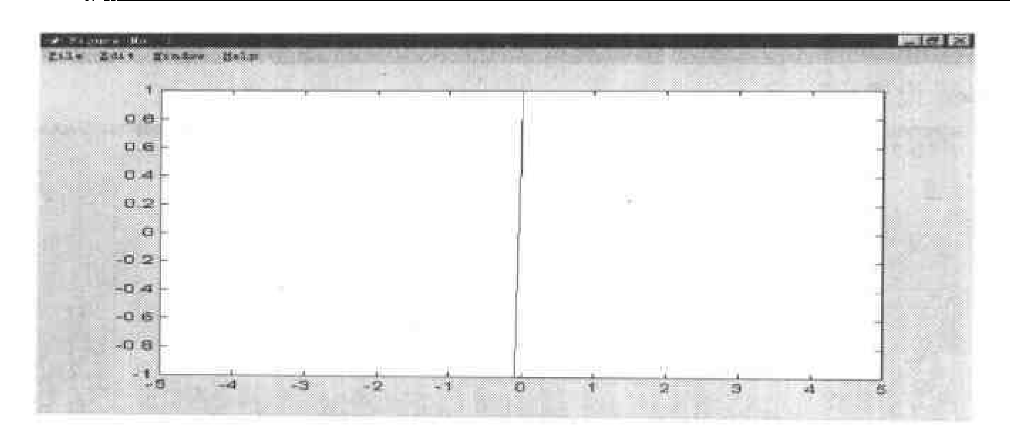


图 6.7 范例 6.12 的运行结果

5. *初始化函数*

(1) initlay 神经网络某一层的初始化函数.

格式：net = initlay (net)

info = initlay (code)

说明：initlay 是一个网络的初始化函数，它根据 NET. layers {i}. initFcn 设置的初始化函数对第 i 层的权进行初始化.

initlay (net) 的输入为 net，是一个神经网络，该函数返回一个第 i 层更新了的网络.

initlay (code) 函数可按照 code 指定的代码返回一些有用的信息，其中 code 可取为

'pnames' 初始化参数的名称.

'pdefaults' 缺省的初始化参数.

initlay 函数没有任何初始化参数.

网络应用：我们可以通过调用 newp，newlin，newff，newcf 及其他新的网络函数产生一个标准的神经网络，该网络使用 initlay 初始化网络的第 i 层.

为设计一个用 initlay 函数进行初始化的网络，需进行如下设置：

1) 把 NET. initFcn 设为 initlay 函数，此时将 NET. initParam 设为空矩阵，这是因为 initlay 函数没有初始化参数；

2) 设置第 i 层的 NET. layers {i}. initFcn 为一个层的初始化函数（例如该函数可以是 initwb 及 initnw).

可以调用 init 函数来初始化一个网络. 参见 newp 函数及 newlin 函数的初始化实例.

算法：网络的第 i 层是根据 NET. layers {i}. initFcn 来初始化其权值和阈值的.

(2) initwb 神经网络的某一层的权值和阈值的初始化函数.

格式：net = initwb (net, i)

说明：initwb 函数是可以按照其特定的初始化函数对网络某一层的权值和阈值进行初始化.

initlay (net, i) 的输入中 net 代表是一个神经网络，i 表示网络的第 i 层，该函数返回一个第 i 层的权值和阈值都更新了的网络.

网络应用：我们可以通过调用 newp，newlin 函数构造一个标准的神经网络，该网络使用 initwb 来初始化网络的每一层.

为设计一个用initwb 函数进行初始化的网络，需进行如下设置：

1）把NET. initFcn 设为initlay 函数，此时将NET. initParam 设为空矩阵，这是因为initlay 函数没有初始化参数；

2）设置第i 层的NET. layers {i}. initFcn 为initwb 函数；

3）设置第i 层的NET. inputWeights {i, j}. learnFcn 函数为一权值初始化函数，设置第i 层的NET. layerWeights{i,j}. learnFcn 为一个权值初始化函数，设置NET. biases {i}. learnFcn 为第i 层的阈值初始化函数.（例如该函数可以是rands 及midpoint）.

可以调用init 函数来初始化一个神经网络. 可参见newp 函数及newlin 函数的例子中的训练部分.

算法：网络的第i 层的每一个权值（阈值）是根据其权值（阈值）初始化函数计算出的.

（3）initzero 将权值设置为零的初始化函数

格式：W = initzero (S, PR)

b = initzero (S, [1 1])

说明：initzero (S, PR) 函数是将权值设置为零的初始化函数，其中包含神经元个数S 及输入向量的范围PR. 该函数返回一个零权值矩阵.

initzero (S, [1 1]) 函数是将阈值设置为零的初始化函数，它能返回一个为0 阈值向量.

例6.13 对于2 个输入（其范围为：[0 1] 及 [−2 2]）5 个神经元的神经网络的某一层，计算其权值和阈值的初值.

W = initzero (5, [0 1; −2 2])

b = initzero (5, [1 1])

网络应用：我们可以通过调用newp，newlin 函数构造一个标准的神经网络，该网络使用initzero 初始化网络的权值.

为设计一个用midpoint 函数进行初始化网络的权值和阈值，需进行如下设置：

1）把NET. initFcn 设为initlay 函数，此时将NET. initParam 自动设为initlay 的默认参数；

2）设置第i 层的NET. layers{i}. initFcn 为initwb 函数；

3）设置第i 层的NET. inputWeights{i,j}. learnFcn 函数为initzero 函数，设置第i 层的NET. layerWeights{i,j}. learnFcn 为一个initzero 函数，设置第i 层的NET. biases {i}. learnFcn 为initzero 函数.

可以调用init 函数来初始化一个网络. 可参见newp 函数及newlin 函数的初始化部分.

6. 性能函数

mae 平均绝对误差性能函数.

格式：perf = mae(e,x,pp)

perf = mae(e,net,pp)

info = mae(code)

说明：mae 是一个求网络的性能参数的函数. mae (e,x,pp) 函数可采用 1 至 3 个参数：

e　误差向量矩阵或向量.

x　所有的权值和阈值向量(可忽略).

pp　性能参数(可忽略).

该函数执行后可返回平均绝对误差. 误差 E 可以是元数组或矩阵：

e　Nt×TS 元数组，每一个元素 E{i,ts}是一个 Q 组 Vi 维的矩阵或空矩阵[].

e　Q 组 Vi 总和维的矩阵.

其中

Nt = net. numTargets

TS = 时间步长，

Q = 输入向量的个数，

Vi = net. targets{i}. size

mae (e, net, pp) 函数形式与 mae (e, x, pp) 形式的不同之处是用 net 取代了 x，net 是由 x 得到的网络. 该函数的参数中只有 NET 是取代了 x 的神经网络(可忽略)，其余参数与 mae (e, x, pp) 相同.

info = mae (code) 中 mae (code) 返回对每一个 code 字符串有用的信息：

'deriv'　导出函数的返回名称.

'name'　返回全名.

'pnames'　返回训练参数的名称.

'pdefaults'　返回缺省的训练参数.

例 6.14　建立一个输入的单个神经元的感知器，输入范围是 [−10 10]，下面给出网络的输入和目标向量：

```
p = [-10 -5 0 5 10];
t = [0 0 1 1 1];
```

求出网络平均绝对误差.

```
net = newp ( [-10 10], 1);
p = [-10 -5 0 5 10];
t = [0 0 1 1 1];
a = sim (net, p)
e = t-a
perf = mae (e)
```

该程序执行结果为

```
a = [1 1 1 1 1]
e = [-1 -1 0 0 0]
perf = 0.4000
```

mae 可以只使用一个参数，而其他的参数是可省略的.

网络的应用：可以用 newp 函数，并使用 mae 来构造一个标准网络.

为了设计一个用 mae 函数进行训练特定的网络，需进设置 NET. performFcn 为

mae. 当 mae 没有性能参数时，它将自动设置 NET. performParam 为空矩阵 [].

在上述任一种情况下，调用 train 和 adapt 函数将在 mae 中计算网络的性能参数. 可参见 newp 的例子.

7. 学习函数

(1) learnp 感知器的权值/阈值学习函数

格式：[dW,LS] = learnp(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)

[db,LS] = learnp(b,ones(1,Q),Z,N,A,T,E,gW,gA,D,LP,LS)

info = learnp(code)

说明：learnp 函数是感知器的权值/阈值学习函数.

learnp (W, P, Z, N, A, T, E, gW, gA, D, LP, LS) 使用下列输入参数：

W　　S×R 维的权值矩阵（或 S×1 维的阈值向量）.

P　　Q 组 R 维的输入向量（或 Q 组单个输入）.

Z　　Q 组 S 维的权值输入向量.

N　　Q 组 S 维的网络输入向量.

A　　Q 组 S 维的输出向量.

T　　Q 组 S 维的目标向量.

E　　Q 组 S 维的误差向量.

gW　　S×R 维的性能参数的梯度.

gA　　Q 组 S 维的性能参数的输出梯度.

LP　　学习参数，若没有学习参数，LP = [].

LS　　学习状态，初始值为 [].

该函数可返回以下参数：

dW　　S×R 维的权值（或阈值）变化阵.

LS　　新的学习状态.

learnp (code) 对于每一个 code 代码返回相应的有用的信息：

'pnames'　　返回学习参数的名称.

'pdefaults'　　返回默认的学习参数.

'needg'　　如果该函数使用 gW 或 gA，则返回值为 1.

例 6.15 对具有 2 个输入 3 个神经元的某一层进行权值学习，我们定义一个随机的输入 P 和误差 E.

```
p = rand (2, 1);
e = rand (3, 1)
```

因为 learnp 只需要一些值来计算权值的变化（参见下面的算法），我们将这样学习：

```
dW = learnp ( [], p, [], [], [], [], e, [], [], [], [], [])
```

网络的应用：可以用具有学习函数为 learnp 的 newp 函数建立一个标准的网络.

为了训练一个特定的网络的第 i 层的权值和阈值使用 learnp 函数来学习，需进行如

下设置：

1）设置 NET. trainFcn 为 trainwb.（NET. trainParam 将自动设为 trainwb 的缺省值.）

2）设置 NET. adaptFcn 为 adaptwb.（NET. trainParam 将自动设为 trainwb 的缺省值.）

3）设置第 i 层的 NET. inputWeights{i,j}. learnFcn 为 learnp 学习函数. 同样，设置第 i 层的 NET. layerWeights{i,j}. learnFcn 为 learnp 学习函数. 设置第 i 层的 NET. biases{i}. learnFcn 为一个 learnp 学习函数.（因为学习函数 learnp 没有学习参数，所以网络的权值和阈值的学习参数将自动地被设为空矩阵.）

为训练一个网络(或使其能进行自适应)，需：

1）设置 NET. trainParam（NET. adaptParam）的属性为希望值.

2）调用 train（或 adapt）函数.

可参考 newp 函数的适应和训练的实例.

算法：learnp 函数用神经元的输入 P 及误差 E 根据感知器的学习规则计算一个给定的神经元的权值变化 dW. 学习规则如下：

如果 e = 0，　则 dw = 0

如果 e = 1，　则 dw = p′

如果 e = −1，　则 dw = −p′

上述关系可表达为

dw = e * p′

(2) learnpn 标准化感知器的权值/阈值学习函数

格式：[dW，LS] = learnpn（W，P，Z，N，A，T，E，gW，gA，D，LP，LS）

info = learnpn (code)

说明：learnpn 函数是感知器的权值/阈值学习函数. 当输入样本中具有奇异的向量时，该学习函数能够产生较快的学习速度.

learnpn（W，P，Z，N，A，T，E，gW，gA，D，LP，LS）使用下列输入参数：

W　S×R 维的权值矩阵（或 S×1 维的阈值向量）.

P　Q 组 R 维的输入向量（或 Q 组单个输入）.

Z　Q 组 S 维的权值输入向量.

N　Q 组 S 维的网络输入向量.

A　Q 组 S 维的输出向量.

T　Q 组 S 维的目标向量.

E　Q 组 S 维的误差向量.

gW　S×R 维的性能参数的梯度.

gA　Q 组 S 维的性能参数的输出梯度.

LP　学习参数，若没有，LP = [].

LS　学习状态，初始值为 [].

函数可返回以下参数：

dW SxR 维的权值（或阈值）变化阵.

LS 新的学习状态.

learnpn (code) 对于每一个 code 代码返回相应的有用信息：

'pnames' 返回学习参数的名称.

'pdefaults' 返回默认的学习参数.

'needg' 如果该函数使用 gW 或 gA，则返回值为 1.

例 6.16 对具有 2 个输入 3 个神经元的某一层用 learnpn 函数进行学习．我们定义一个随机的输入 P 和误差 E.

```
p = rand (2, 1);
e = rand (3, 1);
```

因为 learnpn 只需要一些值来计算权值的变化（参见下面的算法），我们将这样学习：

```
dW = learnpn ( [], p, [], [], [], [], e, [], [], [], [])
```

网络的应用：可以用具有学习函数为 learnpn 的 newp 函数建立一个标准的网络.

为了训练一个特定网络的第 i 层的权值和阈值使用 learnpn 函数来学习，需进行如下设置：

1）设置 NET. trainFcn 为 trainwb.（NET. trainParam 将自动设为 trainwb 的缺省值.）

2）设置 NET. adaptFcn 为 adaptwb.（NET. trainParam 将自动设为 trainwb 的缺省值.）

3）设置第 i 层的每个 NET. inputWeights{i,j}. learnFcn 为一个 learnpn 学习函数. 同样，设置第 i 层的每个 NET. layerWeights{i,j}. learnFcn 为一个 learnpn 学习函数. 设置第 i 层的 NET. biases{i}. learnFcn 为一个 learnpn 学习函数.（因为学习函数 learnpn 没有学习参数，所以网络的权值和阈值的学习参数将自动地被设为空矩阵.）

为训练一个网络（或使其能适应），需：

1）设置 NET. trainParam（NET. adaptParam）的属性为希望值.

2）调用 train（或 adapt）函数.

可参考 newp 函数的适应和训练的实例.

算法：根据感知器的学习规则，learnpn 函数用神经元的输入 P 及误差 E 计算一个给定的神经元的权值变化 dW. 学习规则如下：

pn = p / sqrt (1 + p (1) ^ 2 + p (2) ^ 2) + ... + p (R) ^ 2)

如果 e = 0，则 dw = 0

如果 e = 1，则 dw = pn'

如果 e = −1，则 dw = −pn'

上述关系可表达为

dw = e * pn'

8. 自适应函数

adaptwb 网络的权值和阈值的自适应函数.

格式：[net, Ac, El] = adaptwb (net, Pd, T, Ai, Q, TS)

info = adaptwb (code)

说明：adaptwb 函数是一个网络的自适应函数，该函数可以根据其学习函数来更新网络的权值和阈值. adaptwb 函数的参数为

net　神经网络.

Pd　延迟输入.

Tl　每一层的目标向量.

Ai　初始输入条件.

Q　输入向量的个数.

TS　时间步长.

网络的训练结束之后，具有权值和阈值的学习函数的网络返回下列参数：

net　更新后的网络.

Ac　总的层输出.

El　该层的误差.

网络的自适应是按照adaptwb 的训练参数进行的. 下面给出其缺省值：net. adaptParam. passes 为 1 表示对所有的输入数据训练一次. 这些变量的维数为

Pd　No×Ni×TS 元数组，每一元素 P{i,j,ts} 为一 Zij×Q 矩阵.

Tl　Nl×TS 元数组，每一元素 P{i,ts} 为一 Vi×Q 矩阵或空矩阵 [].

Ai　Nl×LD 元数组，每一元素 Ai{i,k} 为一 Si×Q 矩阵.

Ac　Nl×(LD+TS) 元数组，每一元素 Ac{i,k} 为一 Si×Q 矩阵.

El　Nl×TS 元数组，每一元素 El{i,k} 为一 Si×Q 矩阵或空矩阵[].

其中

Ni = net.numInputs

Nl = net.numLayers

LD = net.numLayerDelays

Ri = net.inputs{i}.size

Si = net.layers{i}.size

Vi = net.targets{i}.size

Zij = Ri * length(net.inputWeights{i,j}.delays)

adaptwb(code) 返回每一code 字符串代表的有用信息.

'pnames'　训练参数的名称.

'pdefaults'　缺省的训练参数.

网络的应用：通过调用 newp 函数或者 newlin 函数，并使用adaptwb 函数，可以产生一个标准网络.

为了设计一个用adaptwb 函数进行自适应的网络，需进行如下设置：

1) 设置 NET.adaptFcn 为 adaptwb. (NET.adaptParam 将自动设为 adaptwb 的缺省值).

2) 设置第 i 层每个的 NET.inputWeights{i,j}.learnFcn 为某一个学习函数. 同样，设置第 i 层每个的 NET.layerWeights{i,j}.learnFcn 为某一个学习函数. 设置第 i 层的

NET. biases{i}. learnFcn 为某一个学习函数.(网络的权值和阈值的学习参数将自动地被设为该学习函数的缺省值.)

为使网络进行自适应,需做如下几步:

1) 设定 NET. adaptParam 为所需的值.

2) 设定权值和阈值学习参数为所需的值.

3) 调用 adapt 函数.

可参见 newp 及 newlin 的 训练举例.

算法:输入序列的每一步结束后,各权值和阈值都会按照学习函数来更新.

9. 训练函数

trainwb 网络的权值和阈值的训练函数.

格式: [net, tr] = trainwb (net, Pd, Tl, Ai, Q, TS, VV)

info = trainwb (code)

说明: trainwb 函数是一个网络训练函数,该函数根据其学习函数更新权值和阈值.
trainwb 函数的输入参数为

net 神经网络.

Pd 延迟输入(反馈).

Tl 每一层的目标向量.

Ai 初始输入条件.

Q 输入向量的数目.

TS 时间步长.

VV 或者是空矩阵 [] 或者是确定的向量的结构.

该函数的返回参数为

net 训练后的网络.

TR 每一步中各个值的训练记录:

TR. epoch 训练次数.

TR. perf 训练性能.

TR. vperf 验证性能.

TR. tperf 测试性能.

训练过程是按照 trainwb 的给定训练参数进行的,下面给出这些参数的缺省值:

net. trainParam. epochs = 100 为最大的训练次数.

net. trainParam. *goal* = 0 训练的目标.

net. trainParam. max_fail = 5 最小的训练次数.

net. trainParam. show = 25 修改显示迭代次数.

net. trainParam. time = inf 每秒钟训练的最大次数.

这些变量的维数是:

Pd No×Ni×TS 元数组,每个元素 Pd {i, j, ts} 是一个 Dij×Q 的矩阵.

Tl Nl×TS 元数组,每个元素 P {i, ts} 是一个 Vi×Q 的矩阵或空矩阵.

Ai　　Nl×LD 元数组，每个元素 Ai {i, k} 是一个 Si×Q 的矩阵.

其中

Ni = net.numInputs

Nl = net.numLayers

LD = net.numLayerDelays

Ri = net.inputs {i} .size

Si = net.layers {i} .size

Vi = net.targets {i} .size

Dij = Ri×length (net.inputWeights {i, j} .delays)

如果 VV 不是空矩阵，则它必须是一个给定的向量：

VV.Pd　　给定的输入延迟.

VV.Tl　　给定的层的目标.

VV.Ai　　给定的初始输入条件.

VV.Q　　给定的输入数据的个数.

VV.TS　　给定的时间步长.

trainwb (code) 返回每一个 code 字符串的有用信息.

'pnames'　　训练参数的名称.

'pdefaults'　　缺省的训练参数.

网络的应用：通过调用 newp 或 newlin 函数并使用 trainwb 函数可以产生一个标准的网络.

为了设计一个用 trainwb 函数进行训练的网络，需进行如下设置：

1) 设置 NET.trainFcn 为 trainwb.（这将使 NET.trainParam 设为 trainwb 的缺省值.）

2) 设置第 i 层的 NET.inputWeights{i,j}.learnFcn 为某个学习函数. 同样，设置第 i 层的 NET.layerWeights{i,j}.learnFcn 为某个学习函数. 设置第 i 层的 NET.biases{i}.learnFcn 为一个学习函数.（网络的权值和阈值的学习参数将自动地被设为该学习函数的缺省值.）

为训练该网络，需做如下几步：

1) 设定 NET. trainParam 为所需的值.

2) 设定权值和阈值学习参数为所需的值.

3) 调用 train 函数.

可参见 newp 及 newlin 函数中的训练举例.

算法：输入序列的每一步结束后，每个权值和阈值都会按照学习函数来更新.

网络训练将在下列情况下终止：

1) 达到最大的训练次数.

2) 目标的性能参数达到最小.

3) 超出最大的时间限制.

6.3　感知器网络设计实例

6.3.1　简单的分类问题

例 6.17　设计单一感知器神经元来解决一个简单的分类问题：将 4 个输入向量分为两类，其中两个输入向量对应的目标值为 1，另两个向量对应的目标值为 0.

输入向量为

P = [−1 −0.5 +0.3 −0.1;
　　−0.5 +0.5 −0.5 +1.0]

目标向量为

T = [1 1 0 0]

首先我们来分析此问题，输入向量可以用图 6.8 描述，与目标值 0 相对应的输入向量用符号"o"表示，与目标值 1 对应的输入向量用符号"+"表示.

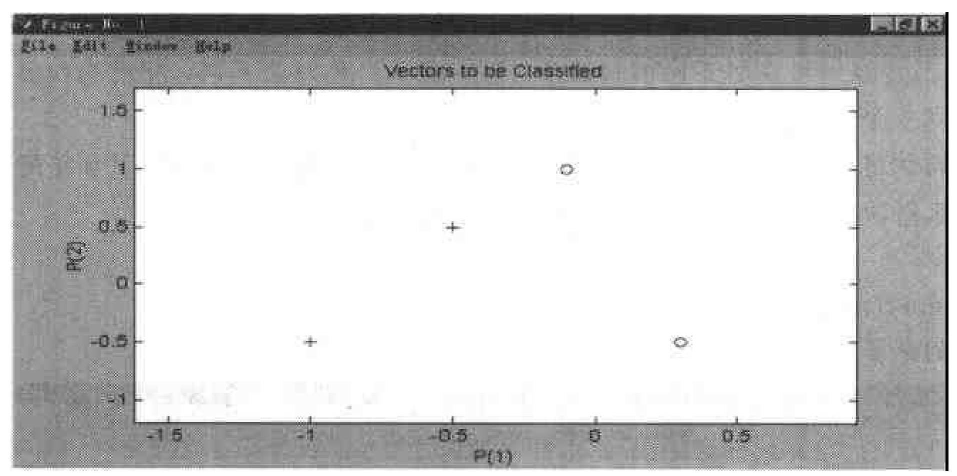


图 6.8　输入向量图

对于这个简单的分类问题，所采用的是两个输入的单个感知器神经元，其结构如图 6.9 所示.

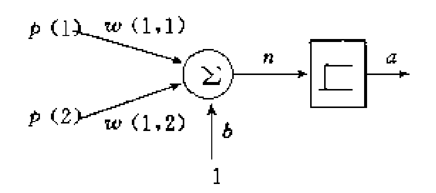


图 6.9　感知器网络结构图

感知器是采用硬限幅阈值函数 hardlim 的神经元.

为设计这个问题，首先用函数 newp 构造一个输入向量均在 [−1，1] 之间的单个神经元感知器：

```
net=newp ( [−1 1; −1 1], 1);
```

用 init 对网络进行初始化，

net=init (net);

然后，利用函数 adapt 调整网络的权值和阈值，直到误差为 0 时结束训练，

[net, Y, E] =adapt (net, P, T);

训练结束后可得到图 6.10 所示的分类曲线. 由图可见分类线将两类输入向量正确地分类.

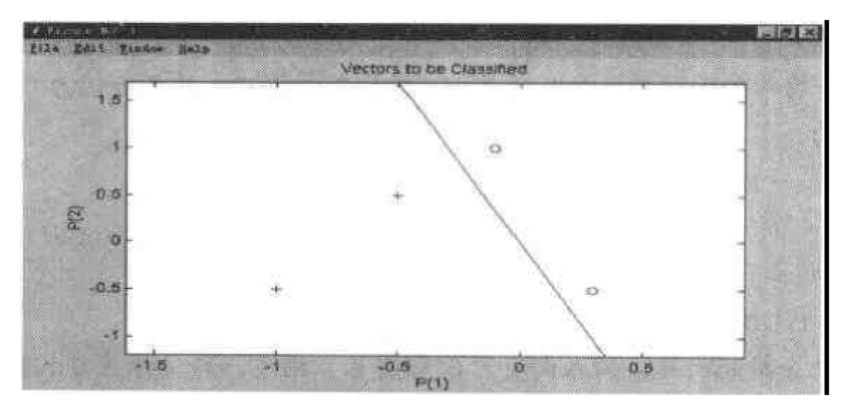


图 6.10 分类结果曲线

感知器神经网络在取不同的初始条件时，其训练的结果不同，即用感知器求解分类问题可得到多个解.

当网络训练完成之后，其权值和阈值就不再改变，这时就可以利用训练好的感知器神经网络来解决实际的分类问题，此时用函数来实现：

p= [0.7; 1.2];

a = sim (net, p);

得到的分类结果如图 6.11 所示.

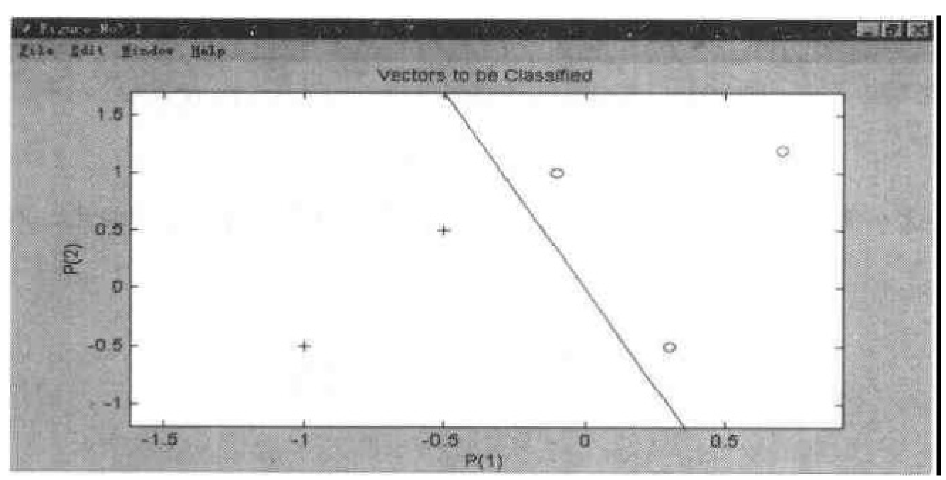


图 6.11 感知器应用结果

由图 6.11 可见该感知器网络能对输入向量进行正确地分类，同时验证了网络的正确性.

下面给出这一分类问题的 MATLAB 程序：

% percept1. m

% NEWP 建立一个感知器.

```
% SIM      对感知器神经网络进行仿真.
% ADAPT    修正感知器神经网络.
% P 为输入向量
P = [ -1 -0.5 -0.3 -0.1;
      0.5 +0.5 -0.5 +1.0];
% T 为目标向量
T = [1 1 0 0];
% 绘制输入向量
plotpv (P, T);
pause % 按任意键继续
% 建立一个感知器
net=newp ( [-1 1; -1 1], 1);
watchon;
cla;
% 绘制输入向量
plotpv (P, T);
% 绘制分类线
linehandle=plotpc (net. IW {1}, net. b {1});
E=1;
% 初始化感知器
net=init (net);
linehandle=plotpc (net. IW {1}, net. b {1});
% 修正感知器网络
while (sse (E))
[net, Y, E] =adapt (net, P, T);
linehandle=plotpc (net. IW {1}, net. b {1}, linehandle);
drawnow;
end;
pause
watchoff;
% 利用训练好的感知器进行分类
p= [0.7; 1.2];
a = sim (net, p);
plotpv (p, a);
ThePoint=findobj (gca,'type','line');
set (ThePoint,'Color','red');
hold on;
plotpv (P, T);
plotpc (net. IW {1}, net. b {1});
```

```
hold off;
disp ('End of percept1')
```

6.3.2　多个感知器神经元的分类问题

例 6.18　将上例中的输入向量扩充为 10 组，将输入向量分为 4 类，即输入向量为

```
P =[0.1 0.7 0.8 0.8 1.0 0.3 0.0 -0.3 -0.5 -1.5;
    1.2 1.8 1.6 0.6 0.8 0.5 0.2 0.8 -1.5 -1.3];
```

所对应的目标向量为

```
T = [1 1 1 0 0 1 1 1 0 0;
     0 0 0 0 0 1 1 1 1 1];
```

输入向量可以用图 6.12 描述，与目标值 00 相对应的输入向量用符号"o"表示，与目标值 01 对应的输入向量用符号"*"表示，与目标值 10 对应的输入向量用符号"+"表示与目标值 11 对应的输入向量用符号"×"表示．

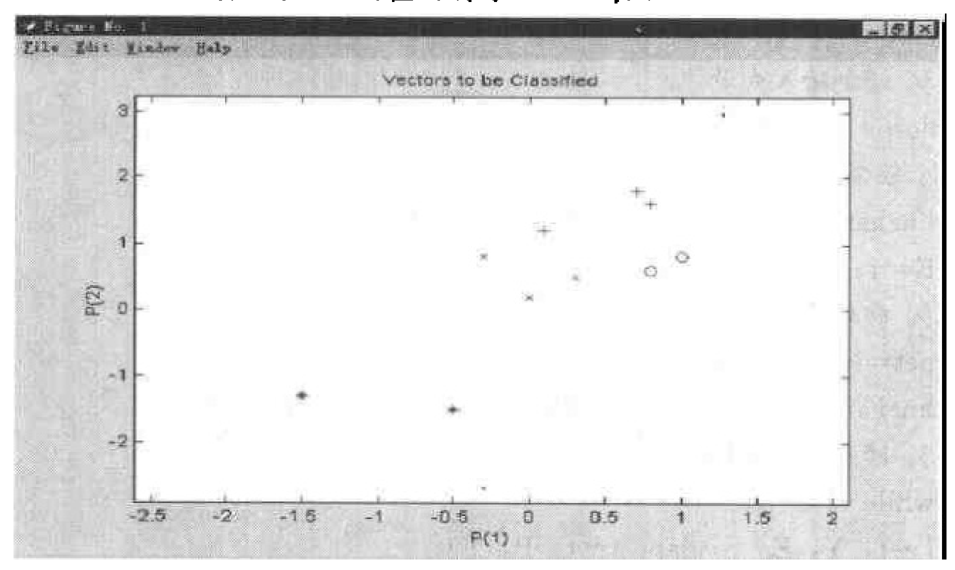


图 6.12　输入向量图

本例所采用的网络具有两个输入，两个神经元，网络结构如图 6.13 所示．

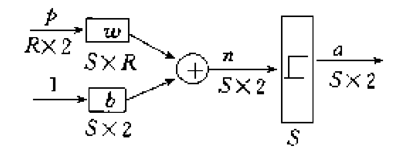


图 6.13　网络结构

首先用函数 newp 构造一个输入向量均在 [−1，1] 之间的两个神经元感知器，

```
net=newp ([-1 1; -1 1], 2);
```

用 init 对网络进行初始化，

```
net=init (net);
```

然后，利用函数 adapt 调整网络的权值和阈值，直到误差为 0 时结束训练，

```
[net, Y, E] =adapt (net, P, T);
```

经过 10 次权值和阈值的修正，网络将输入向量分成期望的四类，可得到图 6.14 所示的分类曲线. 由图可见分类线将四类输入向量正确地分类.

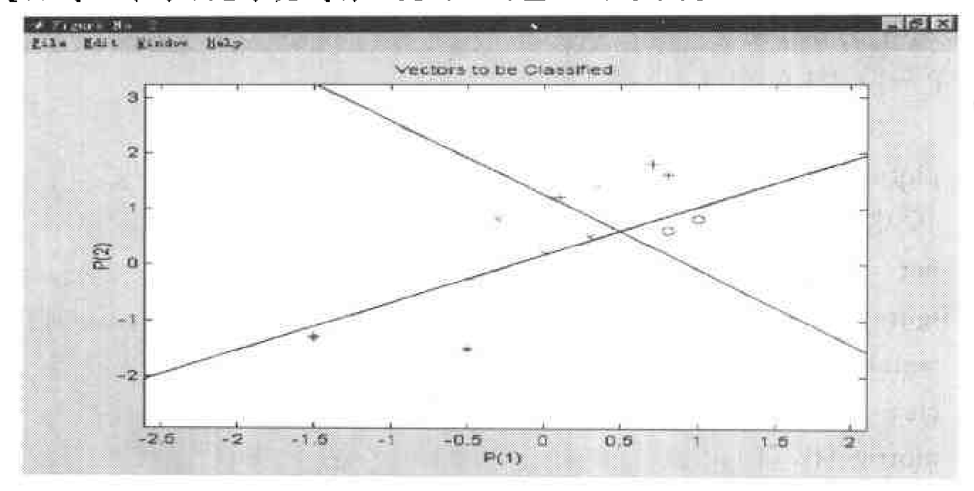


图 6.14 分类结果曲线

当网络训练完成之后，就可以利用训练好的感知器神经网络来解决实际的分类问题，此时用函数来实现：

p= [0.7; 1.2];

a = sim (net, p);

得到的分类结果如图 6.15 所示. 由此可见，该网络可以对输入实现正确分类，从而验证了网络的正确性.

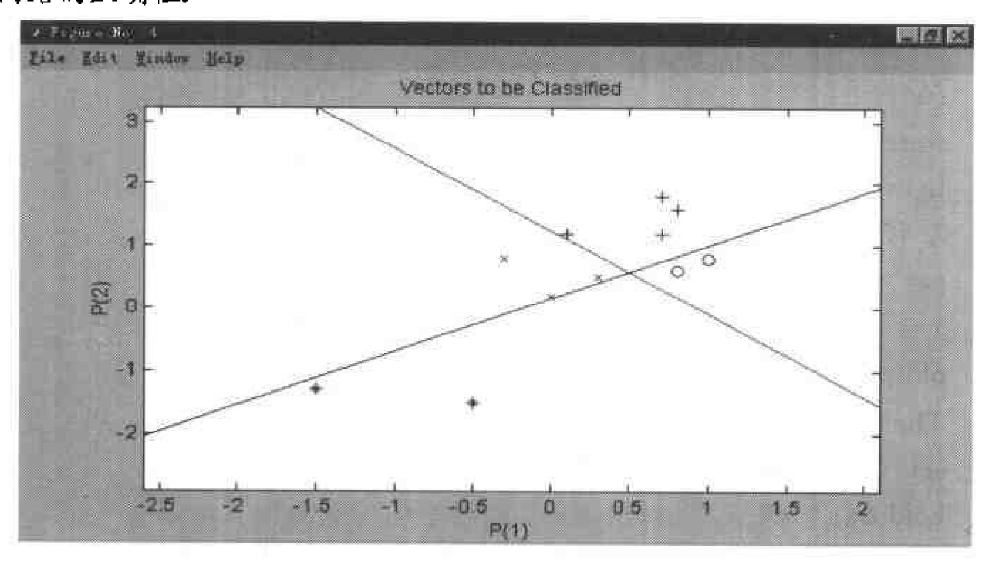


图 6.15 感知器应用结果

下面给出源程序：

```
% percept2. m
% NEWP        建立一个感知器.
% SIM         对感知器神经网络进行仿真.
% ADAPT       修正感知器神经网络.
% P 为输入向量
```

```
P =[0.1 0.7 0.8 0.8 1.0 0.3 0.0 -0.3 -0.5 -1.5;
    1.2 1.8 1.6 0.6 0.8 0.5 0.2 0.8 -1.5 -1.3];
%T 为目标向量
T =[1 1 1 0 0 1 1 1 0 0;
    0 0 0 0 0 1 1 1 1 1];
plotpv (P, T);
%建立一个感知器神经网络
net=newp ( [-1.5 1; -1.5 1], 2);
figure;
watchon;
cla;
plotpv (P, T);
linehandle=plotpc (net.IW {1}, net. b {1});
%调整网络参数
E=1;
net=init (net);
linehandle=plotpc (net. IW {1}, net. b {1});
while (sse (E))
[net, Y, E] =adapt (net, P, T);
linehandle=plotpc (net. IW {1}, net. b {1}, linehandle);
drawnow;
end;
watchoff;
figure;
%利用训练好的感知器网络进行分类
p= [0.7; 1.2];
a = sim (net, p);
plotpv (p, a);
ThePoint=findobj (gca,'type','line');
set (ThePoint,'Color','red');
hold on;
plotpv (P, T);
plotpc (net. IW {1}, net. b {1});
hold off;
disp ('End of percept2')
```

6.3.3 输入奇异样本对网络训练的影响

例 6.19 当网络的输入样本中存在奇异的样本（即该样本向量与其他所有的样本向量比较起来特别大或特别小），此时网络训练的时间将大大增加．我们以下例来分析．

给出输入向量及目标向量：

P = [−0.5 −0.5 +0.3 −0.1 −40; −0.5 +0.5 −0.5 +1.0 +50];

T = [1 1 0 0 1];

同理可得到输入向量分布情况，如图 6.16 所示、与目标值 0 相对应的输入向量用符号"o"表示，与输入向量 1 对应的输入向量用符号"+"表示.

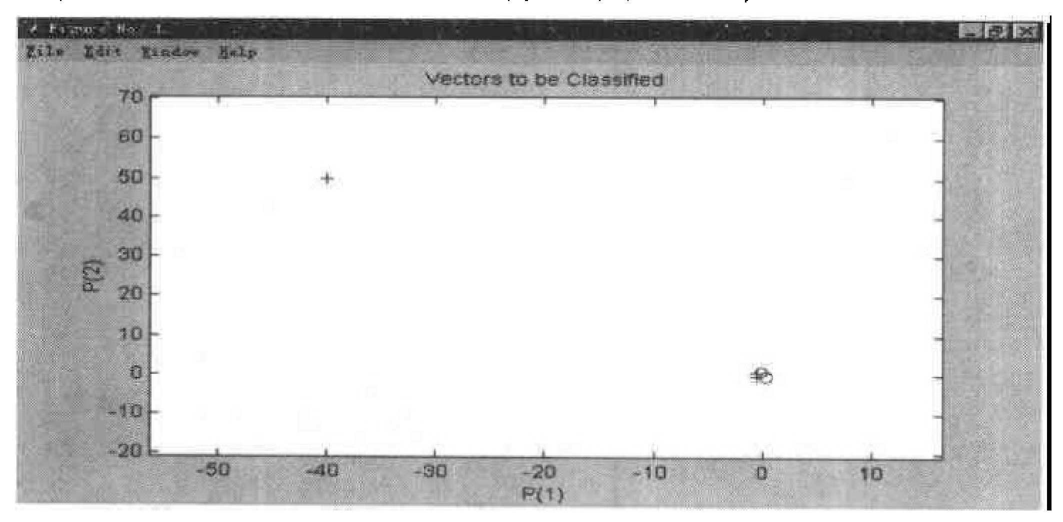


图 6.16 输入向量图

从图中可见，输入向量范围 [−40; 50] 明显地不同于其他输入样本向量. 从仿真过程可见训练的时间大大增加了. 分类的结果如图 6.17 所示.

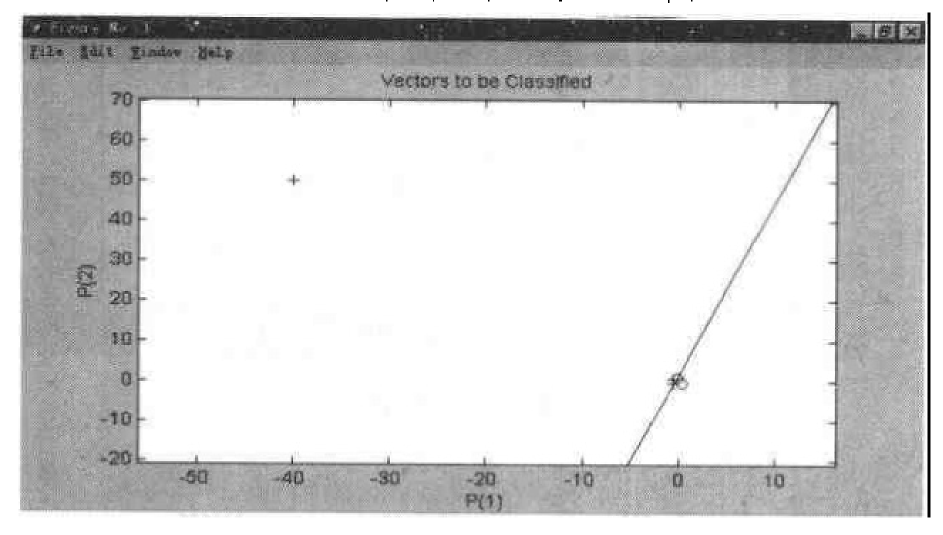


图 6.17 分类结果

用一组输入验证网络的性能，测试的结果如图 6.18 所示. 为了使图形清晰，修改坐标后重新绘制分类线如图 6.19 所示.

由该程序的仿真结果可见，网络可以对输入向量正确分类，但是由于有奇异样本，网络的训练时间大大加长了.

下面给出源程序：

```
% percept3.m
% NEWP        建立一个感知器.
```

```
% SIM        对感知器神经网络进行仿真.
% ADAPT      修正感知器神经网络.
P = [−0.5 −0.5 +0.3 −0.1 −40;
     −0.5 +0.5 −0.5 +1.0 50];
```

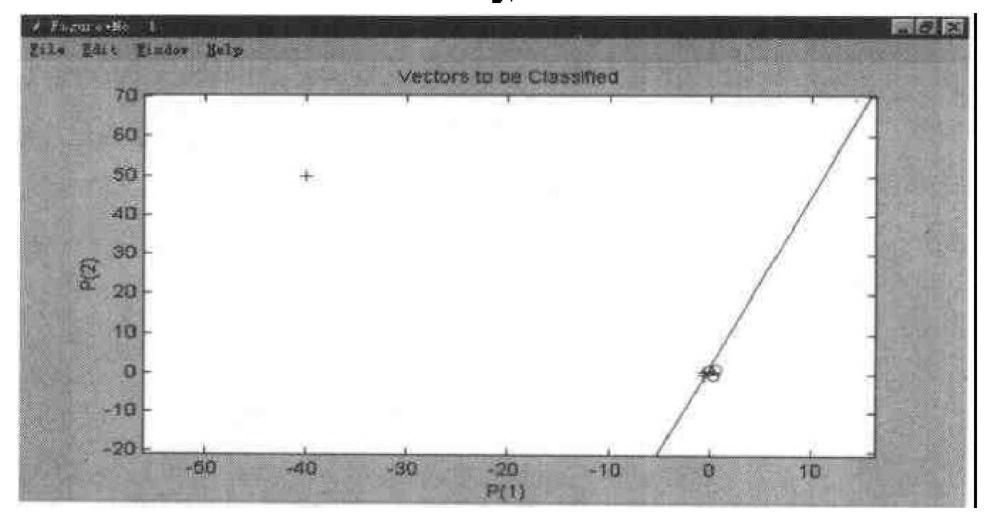


图 6.18　测试结果

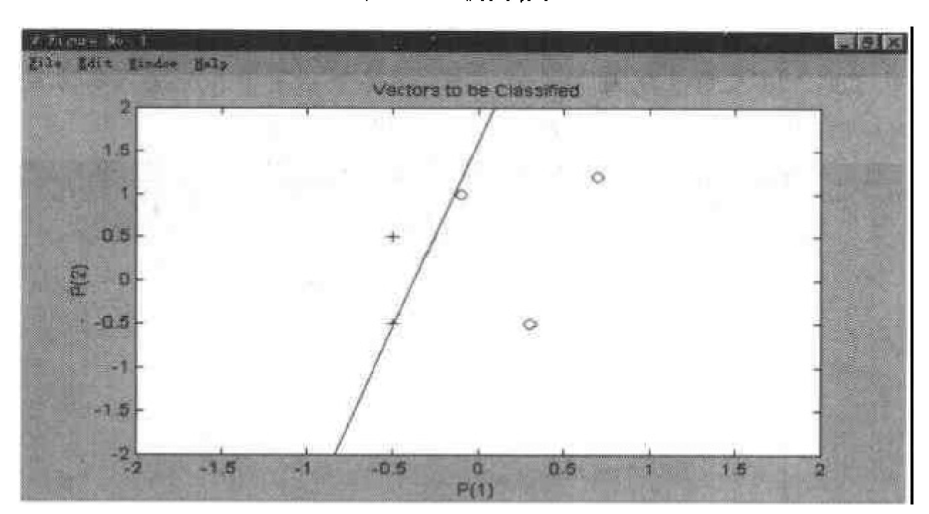


图 6.19　测试结果

```
T = [1 1 0 0 1];
plotpv (P, T);
net=newp ( [−40 1; −1 50], 1);
pause
plotpv (P, T);
linehandle=plotpc (net. IW {1}, net. b{1});
cla;
plotpv (P, T);
linehandle=plotpc (net.IW {1}, net.b{1});
E=1; net. adaptParam. passes=1;
```

```
net=init (net);
linehandle=plotpc (net. IW {1}, net. b{1});
while (sse(E))
    [net,Y,E]=adapt(net,P,T);
    linehandle=plotpc(net.IW{1},net.b{1},linehandle);
    drawnow;
end;
pause
p = [0.7; 1.2];
a = sim (net, p);
plotpv (p, a);
ThePoint=findobj (gca,'type','line');
set (ThePoint,'Color','red');
hold on;
plotpv (P, T);
plotpc (net. IW{1},net.b{1});
hold off;
pause
axis ( [-2 2 -2 2]);
disp ('End of percept3')
```

6.3.4 标准化感知器学习规则

例6.20 用标准化感知器学习规则训练上述例子中的网络.

从上述例子中可见，当输入样本中存在奇异样本，网络的训练时间将大大增加. 为解决此问题，提出一种改进的感知器学习规则——标准化感知器学习规则.

原始的感知器学习规则中权值调整是采用下式：

$$\Delta w = (t - a)p^T = ep^T$$

从上式可见，输入向量 p 越大，权值的变化就越大. 当存在奇异样本时，相对很小的样本需要花很多时间才能同奇异样本所对应的权值变化相匹配. 因此，标准化感知器学习规则试图使奇异样本和其他样本对权值的变化值的影响均衡，即

$$\Delta w = (t-a)\frac{p^T}{\| p \|} = e\frac{p^T}{\| p \|}$$

在神经网络工具箱中，标准化感知器学习规则是由函数 learnpn ( ) 实现的.

标准化感知器学习规则相对于原始感知器学习规则来说，网络的训练时间稍长，但是对于含有奇异样本的输入向量时，标准化感知器学习规则就是非常有效的. 例如上例中采用原始感知器学习规则 learnp ( ) 训练网络，需经过 32 步训练，网络才可达到误差指标，而采用 learnpn ( ) 训练网络，需经过 4 步训练网络就可满足要求.

下面给出源程序：

```
% percept4.m
```

```
% NEWP      建立一个感知器.
% SIM       对感知器神经网络进行仿真.
% ADAPT     修正感知器神经网络.
P =[ -0.5 -0.5 +0.3 -0.1 -40;
    -0.5 +0.5 -0.5 +1.0 50];
T = [1 1 0 0 1];
plotpv(P,T);
pause
net=newp([-40 1;-1 50],1,'hardlim','learnpn');
cla;
plotpv(P,T);
linehandle=plotpc(net.IW{1},net.b{1});
E=1; net.adaptParam.passes=1;
net=init(net);
linehandle=plotpc(net.IW{1},net.b{1});
while (sse(E))
    [net,Y,E]=adapt(net,P,T);
    linehandle=plotpc(net.IW{1},net.b{1},linehandle);
    drawnow;
end;
pause
p = [0.7; 1.2];
a = sim(net,p);
plotpv(p,a);
ThePoint=findobj(gca,'type','line');
set(ThePoint,'Color','red');
hold on;
plotpv(P,T);
plotpc(net.IW{1},net.b{1});
hold off;
pause
axis([-2 2 -2 2]);
disp('End of percept4')
```

6.3.5 线性不可分的输入向量

单层感知器的一个致命的弱点是输入向量必须是线性可分的. 如果输入向量线性可分，感知器即可给出正确的分类结果，否则，对于线性不可分的输入向量，感知器就不能对输入向量进行正确分类. 下面举例说明.

例 6.21 定义如下输入向量：

P =[−0.5 −0.5 +0.3 −0.1 −0.8;
−0.5 +0.5 −0.5 +1.0 +0.0];
T = [1 1 0 0 0];

用感知器对上述输入进行分类.

首先给出输入向量如图 6.20 所示. 试验中给出的最大训练次数为 25，网络训练的结果如图 6.21 所示. 由此结果可见单层感知器无法对输入向量进行正确分类. 即使增加训练次数也得不到正确的分类线，因为输入向量是线性不可分的. 这一问题用单层感知器是无法求解的，要解决此问题需采用多层网络.

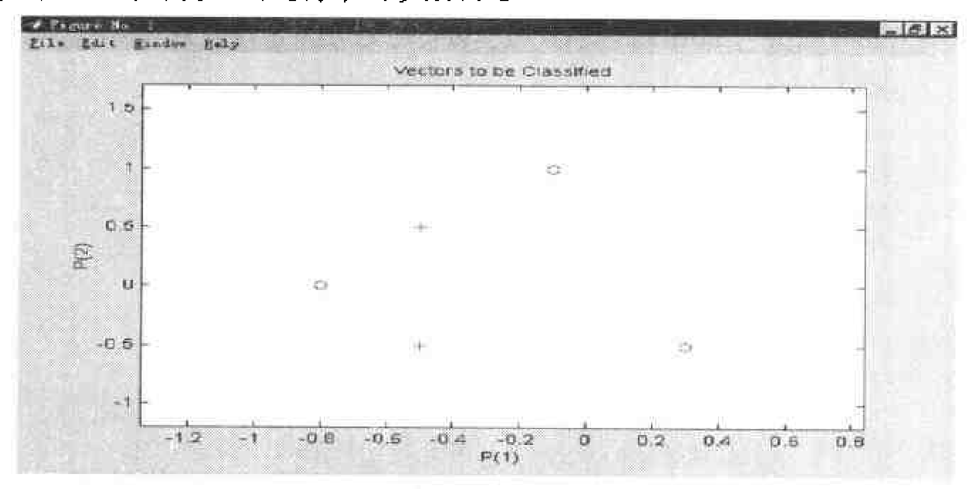


图 6.20 输入向量图

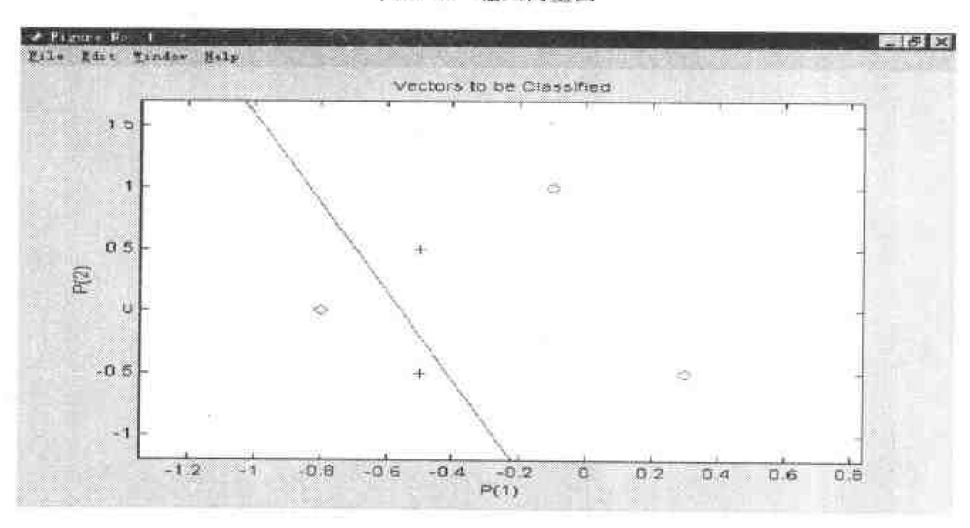


图 6.21 分类情况

下面给出源程序：

```
% percept5.m
% NEWP        建立一个感知器.
% SIM         对感知器神经网络进行仿真.
% ADAPT       修正感知器神经网络.
P = [−0.5 −0.5 +0.3 −0.1 −0.8;
     −0.5 +0.5 −0.5 +1.0 +0.0];
```

```
T = [1 1 0 0 0];
plotpv(P,T);
net=newp([-1 1; -1 1],1);
plotpv(P,T);
linehandle=plotpc(net.IW{1},net.b{1});
pause
for a=1:25
    [net,Y,E]=adapt(net,P,T);
    linehandle=plotpc(net.IW{1},net.b{1},linehandle);
    drawnow;
end;
```

第七章　线性神经网络

线性神经网络是最简单的一种神经元网络，它可以由一个或多个线性神经元构成. 50年代末由威德罗(Widrow)和霍夫(Hoff)提出的自适应线性元件(Adaptive Linear Element，简称 Adaline)是线性神经网络最早的典型代表. 线性神经网络与感知器神经网络的不同之处在于其每个神经元的传递函数为线性函数，因此线性神经网络的输出可以取任意值，而感知器神经网络的输出只能是 0 或 1. 线性神经网络可以采用 Widrow-Hoff 学习规则，也称为 LMS(Least Mean Square)算法来调整网络的权值和阈值. 线性神经网络的学习算法比感知器网络的学习算法的收敛速度和精度都有较大的提高.

线性神经网络主要用于函数逼近、信号处理滤波、预测、模式识别和控制等方面. 本章所讨论的内容是在 MATLAB5.3 基础上进行的.

7.1　线性神经网络原理

7.1.1　线性神经元模型

图 7.1 给出一个线性神经元模型，其传递函数为线性传递函数 purelin，如图 7.2 所示.

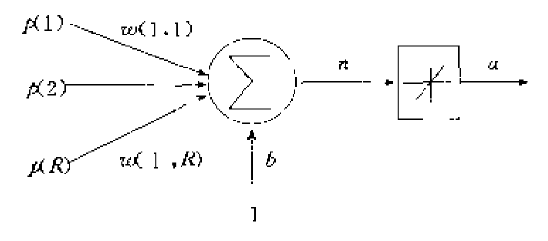


图 7.1　线性神经元模型

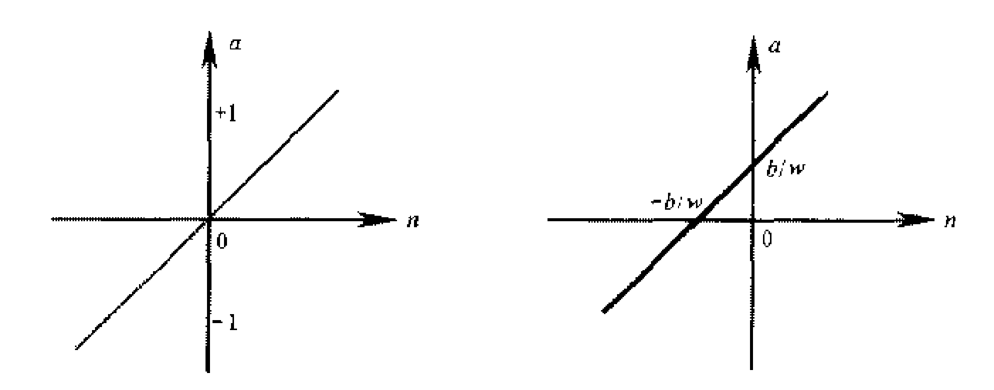


图 7.2　线性传递函数 purelin

由于线性神经网络中神经元的传递函数为线性函数，其输入输出之间是简单的比例关系. 因此，对于单个线性神经元，可通过下式计算出：

$$a=\mathrm{purelin}(w\times p+b)$$

在 MATLAB5.3 中可用

```
net=newlind(P,T);
a=sim(net,P)
```

生成和计算线性神经元的输出.

7.1.2　线性神经网络的模型

图 7.3 给出的是具有 R 个输入的单层(有 S 个神经元)线性神经元网络的两种形式，其权值矩阵为 w，阈值向量为 b，这种网络也称为 Madaline 网络.

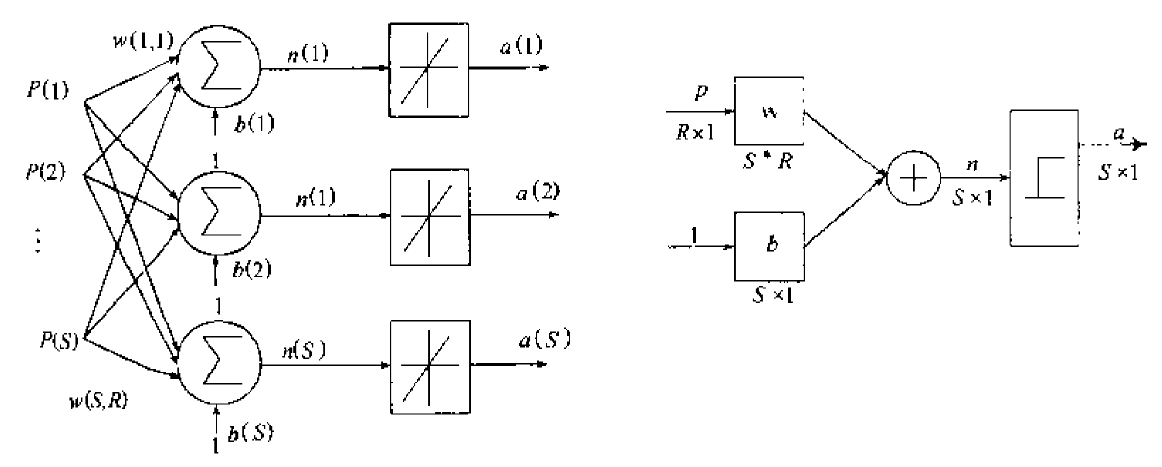


图 7.3　线性神经元网络

Widrow-Hoff 学习规则只能训练单层的线性神经元网络，但这一点并不影响单层线性神经网络的应用，因为对每一个多层线性神经网络而言，都可以设计出一个性能相当的单层线性神经网络.

7.1.3　线性网络的初始化

与感知器网络相同，线性网络的初始化也采用 MATLAB5.3 神经网络工具箱中的 init( )函数. 其用法：

```
net=init(net);
```

这条语句可以将 net 网络权值和阈值设置为 0. MATLAB 神经网络工具箱中函数 newlin 及 newlind 均可产生一个单层线性网络，在这两个函数中使用了 init( )函数，因此得到的线性网络已经是初始化的.

例 7.1　用 newlin 函数设计一个单输入(输入范围是[−1　1])的线性神经网络，学习率为 0.01. 网络设计好之后，要求显示其权值和阈值.

```
net = newlin([-1  1],1,[ ],0.01) ;
net.iw{1}
net.b{1}
```

上述语句执行后得到一个已初始化的线性网络，其权值和阈值均为 0. 对上述网络在给定的输入和目标下进行训练，得到相应的权值和阈值. 如果需要对该网络重新初始化，

即可用

```
net = init(net);
net.iw{1,1}
net.b{1}
```

这样网络的权值和阈值又重新被初始化为0.

7.1.4 线性网络的学习规则

线性网络是采用Widrow-Hoff学习规则,利用learnwh函数来修正网络的权值和阈值.使用Widrow-Hoff学习规则可以用来训练网络某一层的权值和阈值,使其线性逼近一个函数式.下面分析这一规则.

首先定义一个线性网络的误差函数:

$$e(w,b)=\frac{1}{2}[t-a]^2=\frac{1}{2}[t-wp]^2$$

由上式可见,线性网络具有抛物面形的误差曲面,因此只能有一个误差最小值.由于$e(w,b)$只取决于网络的权值和目标矢量,因此,通过调整权值使误差达到最小值.Widrow-Hoff学习规则是通过沿着相对于误差平方和的最快下降方向连续调整网络的权值和阈值.根据梯度下降法,权值矢量的修正正比于当前位置上的$e(w,b)$的梯度,对于第i个输出节点:

$$\begin{aligned}\Delta w(i,j)&=-\eta\frac{\partial e}{\partial w(i,j)}\\&=\eta[t(i)-a(i)]p(j)\end{aligned}$$

或表示为

$$\Delta w(i,j)=\eta\delta(i)p(j)$$
$$\Delta b(i)=\eta\delta(i)$$

其中$\delta z(i)=t(i)-a(i)$.

以上两组表达式即为Widrow-Hoff学习规则,又称为最小均方误差算法(LMS).Widrow-Hoff学习规则的权值变化量正比于网络的输出误差及网络的输入矢量.该算法无需求导数,因此比较简单,又具有收敛速度快和精度高的优点.上述式中η是学习率,当学习率较大时,学习过程加速,网络收敛较快,但是η过大时,学习过程变得不稳定,且误差会加大.因此学习率的取值很关键.神经网络工具函数maxlinlr用于求合适的学习率lr(即η).网络没有阈值时,用lr=maxlinr(P)求学习率;网络有阈值时,用lr=maxlinr(P,'bias')求学习率.

采用Widrow-Hoff规则训练的线性网络,该网络能够收敛的必要条件是被训练的输入矢量必须是线性独立的,且应适当地选择学习率.

7.1.5 线性网络的训练

线性网络的训练过程分如下3步:

1) 根据给定的输入矢量计算网络的输出矢量$a=w\times p+b$,以及与期望输出之间的误差$e=t-a$;

2）将网络输出误差的平方和与期望误差相比较，如果其值小于期望误差，或训练已达到事先设定的最大训练次数，则终止训练，否则继续训练；

3）采用 Widrow-Hoff 学习规则计算新的权值和阈值，并返回到第 1 步.

如果网络训练成功，那么当一个不在训练中的输入矢量输入到网络中时，网络产生一个与此相应的输出矢量. 这种特性被成为泛化功能，这在函数逼近以及输入矢量分类的应用中是很有用的.

如果经过训练的网络不能达到期望目标，可以有两种选择：或检查一下所要解决的问题是否适用于线性网络，或对网络进行进一步的训练.

在 MATLAB5. 3 的神经网络工具箱中，线性网络的训练函数为 adapt. m，adaptwb. m 和 train. m，trainwb. m.

7. 2　有关线性网络的神经网络工具函数

7. 2. 1　MATLAB 中有关线性网络的工具函数

表 7. 1 给出 MATLAB5. 2 中与线性网络相关的神经网络工具函数.

表 7. 1　线性网络的主要神经网络工具函数

函数名称	功　　能
newlind	设计一个线性层
newlin	构造一个线性层
purelin	线性传递函数
dotprod	权值点积函数
netsum	网络输入求和函数
initlay	某层的初始化函数
initwb	某层的权值和阈值的初始化函数
initzero	零权值阈值初始化函数
init	一个网络的初始化函数
mae	求平均绝对误差性能函数
learnwh	Widrow-hoff 的学习规则
adaptwb	网络的权值阈值的自适应函数
adapt	神经网络的自适应函数
trainwb	网络的权值和阈值训练函数
train	神经网络训练函数
maxlinlr	线性层的最大学习率
errsurf	计算误差性能曲面
sim	仿真一个神经网络

7. 2. 2　工具函数详解

对于表 7. 1 给出的函数中有些已在 6. 2 节中做过介绍，此处不在重述，下面分类介绍其余的函数.

1. 网络函数

(1) newlind 设计一个线性层

格式:net = newlind(P,T)

说明:newlind(P,T)的输入参数为

P　R×Q 维的 Q 组输入向量的矩阵.

T　S×Q 维的 Q 组目标分类向量.

该函数返回一个线性层,该线性层是将输入 P 设计为输出 T(具有最小均方误差和).

例 7.2　设一个线性层具有如下的给定输入 P 和输出目标 T

```
P = [1 2 3];
T = [2.0 4.1 5.9];
```

下面我们使用 newlind 函数来设计这样的网络并且检验其输出.

```
P = [1 2 3];
T = [2.0 4.1 5.9];
net = newlind(P,T);
Y = sim(net,P)
```

该程序的仿真结果为

```
Y =
    2.0500    4.0000    5.9500
```

算法:

newlind 函数通过求解下面的线性方程,根据输入 P 和目标 T 计算一个线性层的权值和阈值.

[W b] * [P; ones] = T

(2) newlin 建立一个线性网络

格式:net = newlin(PR,S,ID,LR)

new = newlin

说明:线性层常常在信号处理和预测中用做自适应滤波器. newlin(PR,S,ID,LR)的参数为

PR　R 个输入元素的最大、最小值的矩阵(R×2).

S　输出向量的个数.

ID　输入延迟向量,缺省值= [0].

LR　学习率,缺省值 = 0.01;

该函数可返回一个新的线性层.

net = newlin (PR,S,0,P) 函数用 0,P 取代了参数 ID,LR,其中:

P　输入向量的矩阵.

此时函数返回一个线性层,该线性层具有对于输入 P 而言的最大的稳定学习率.

例 7.3　设计一个单输入(输入范围为[−1　1])单个神经元的线性网络,输入延迟为 0 和 1,学习率为 0.01. 对给定的输入 P1 进行仿真:

P1 = {0 −1 1 1 0 −1 1 0 0 1}

再用给定的目标 T1:

T1 = {0 −1 0 2 1 −1 0 1 0 1}

根据该目标对线性网络进行自适应训练:

```
net = newlin([-1 1],1,[0 1],0.01) ;
P1 = {0 -1 1 1 0 -1 1 0 0 1};
Y = sim(net,P1)
```

仿真结果为

```
Y =
    [0]  [0]  [0]  [0]  [0]  [0]  [0]  [0]  [0]  [0]
```

下面根据目标 T1 对网络进行自适应(由于这是首次调用 adapt ,使用的是缺省的输入延迟条件).

```
T1 = {0 -1 0 2 1 -1 0 1 0 1};
[net,Y,E,Pf] = adapt(net,P1,T1) ;
Y
```

仿真结果为

```
Y =
    [0]  [0]  [0]  [0]  [0.0300]  [-0.0103]  [0.0200]  [0.0395]
[0.0192]  [0.0587]
```

例 7.4 用上面得到的最终的 Pf 作为初始条件,使此线性网络适应一个新的输入:

P2 = {1 0 -1 -1 1 1 1 0 -1},

和目标:

T2 = {2 1 -1 -2 0 2 2 1 0}.

根据设置的输入和目标对线性网络进行训练:

```
P2 = {1 0 -1 -1 1 1 1 0 -1};
T2 = {2 1 -1 -2 0 2 2 1 0};
[net,Y,E,Pf] = adapt(net,P2,T2) ;
Y
```

仿真结果为

```
Y =
    [0.0775]  [0.0872]  [-0.0116]  [-0.0800]  [0.0573]  [0.1924]
[0.2466]  [0.1668]  [-0.0614]
```

例 7.5 对上面的线性网络进行初始化得到新的权值和阈值,用上述的输入 P1 和 P2 训练这个初始化的线性网络使其误差达到 0.1,最大训练次数为 200.

```
net = init(net);
P3 = [P1 P2];
T3 = [T1 T2];
net.trainParam.epochs = 200;
net.trainParam.goal = 0.1;
net = train(net,P3,T3) ;
Y = sim(net,[P1 P2])
```

得到的仿真结果为

```
Y =
```

[0.1869]　[−0.5581]　[0.1638]　[1.7000]　[0.9550]　[−0.5581]　[0.1638]　[0.9550]　[0.1869]　[0.9319]　[1.7000]　[0.9550]　[−0.5581]　[−1.3262]　[0.1638]　[1.7000]　[1.7000]　[0.9550]　[−0.5581]

图 7.4 绘出了训练结果的误差与给定的目标误差曲线.

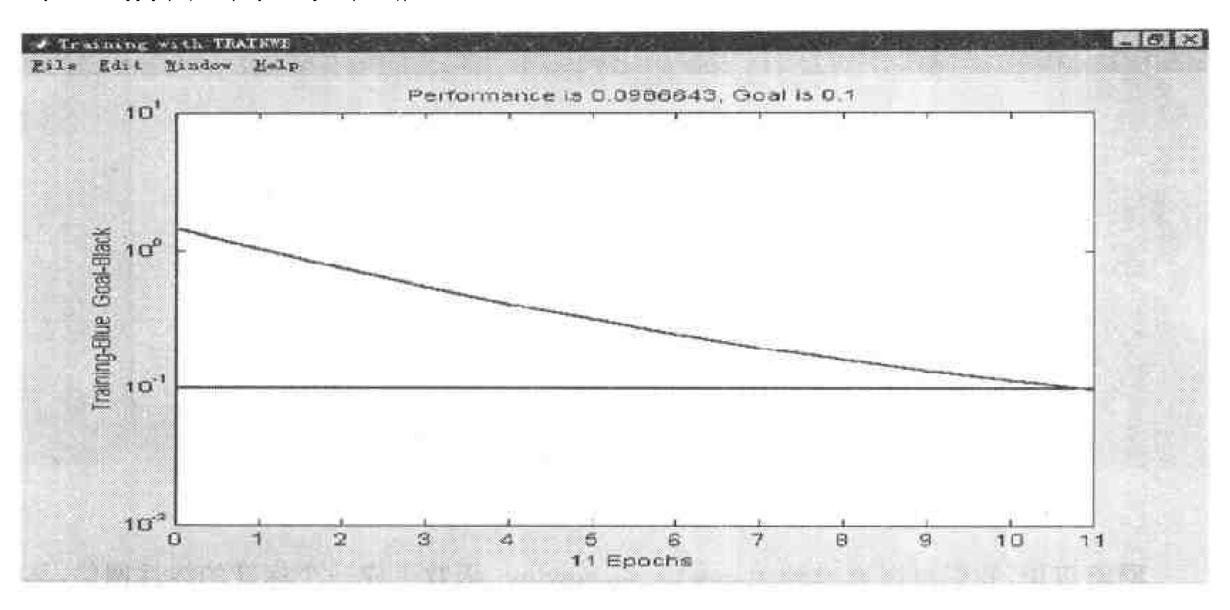


图 7.4　训练结果的误差与给定的目标误差曲线

算法:线性层是一个包含 dotprod 权值函数、netsum 网络输入函数及 purelin 传输函数的单独层. 该层具有输入权值和阈值. 权值和阈值是用 initzero 函数来初始化的. 线性层的适应或训练是用 adaptwb 或 trainwb 函数来完成的. adaptwb 及 trainwb 函数用 learnwh 学习函数修正权值和阈值. 用 mse 计算线性网络的误差性能.

2. 传递函数

purelin 是一个线性传输函数

格式:A = purelin(N)

　　info = purelin(code)

说明:purelin 是一个线性传递函数. 该传递函数根据网络的输入计算线性层的输出.

purelin(N)函数的输入参数为

　　N　　S×Q 是网络输入,

且函数返回 A .

info = purelin (code)函数对每一个 code 代码返回相应的有用信息:

'deriv'　　返回一个导数函数名.

'name'　　返回一个传递函数的全称.

'output'　　返回输出范围.

'active'　　返回传递函数的输入范围.

例 7.6　产生 purelin 传递函数的图形.

```
n = -5:0.1:5;
a = purelin(n);
```

plot(n,a)

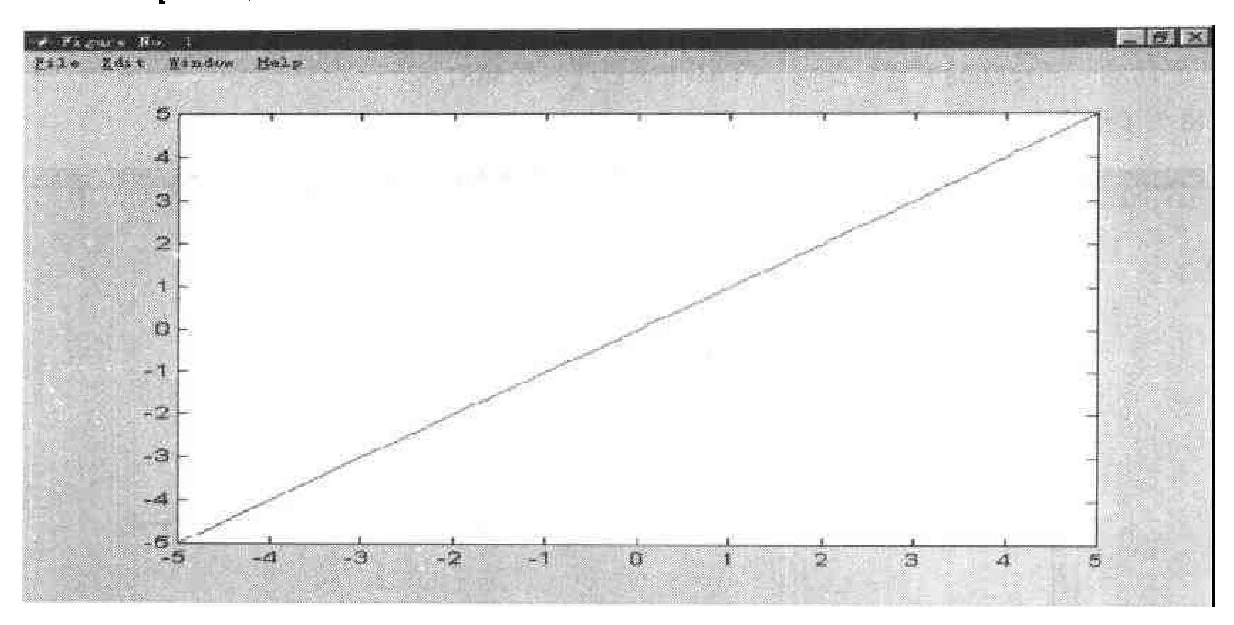


图 7-5 purelin 传递函数

网络应用：我们可以通过调用 newlin 或 newlind 函数生成一个标准的线性网络，其中使用了 purelin 函数. 用函数 purelin 改变网络的某一层，需设置：

NET.layers{i}.transferFcn = purelin.

在上述任一种情况下，调用 sim 来仿真具有 purelin 的网络. 仿真的例子参见 newlin 或 newlind 函数.

算法：purelin(n) = n

3. 学习函数

learnwh 是 Widrow-Hoff 学习函数.

格式：[dW,LS] = learnwh(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)

[db,LS] = learnwh(b,ones(1,Q),Z,N,A,T,E,gW,gA,D,LP,LS)

info = learnwh(code)

说明：learnwh 是 Widrow-Hoff 权值、阈值学习函数，也称为 delta 准则或最小方差准则. 该函数可修改神经元的权值和阈值，使输出误差的平方和最小；可沿着误差平方和的最速下降方向连续调整网络的权值和阈值. 由于线性网络的误差性能表面是抛物面，仅有一个最小值，故能保证网络是收敛的，只要学习率不超出用 maxlinlr 函数计算的最大值.

learnwh(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)函数的参数为

W　S×R 权值矩阵(或 S×1 的阈值向量)，

P　R×Q 输入向量(或 Q 组单个输入)，

Z　S×Q 加权输入向量，

N　S×Q 网络的输入向量，

A　S×Q 输出向量，

T　S×Q 某层的目标向量，

E　　S×Q 某层的误差向量，

gW　S×R 误差性能的梯度，

gA　S×Q 误差性能的输出梯度，

LP　学习参数，若无，LP = []，

LS　学习状态，应初始化为[].

该函数返回：

dW　S×R 权值(或阈值)变化矩阵

LS　新的学习状态.

网络按照 learnwh 的学习参数进行学习，下面给出其缺省值：

LP.lr = 0.01　　学习率

info = learnwh(*code*)函数针对每一种 code 代码返回相应的有用信息：

'pnames'　　返回学习参数名，

'pdefaults'　　返回缺省的学习参数，

'needg'　　若该函数使用 gW 或 gA 则返回 1.

例 7.7　给出一个具有 2 个输入、3 个神经元的神经层，对其定义一个随机输入 P 和误差 E，给出其学习率 lp.lr = 0.5.

```
p = rand(2,1)
e = rand(3,1)
lp.lr = 0.5;
dW = learnwh([],p,[],[],[],[],e,[],[],[],lp,[])
```

上述程序的仿真结果为

```
p =
    0.7621
    0.4565
e =
    0.0185
    0.8214
    0.4447
dW =
    0.0071    0.0042
    0.3130    0.1875
    0.1695    0.1015
```

网络的应用：通过调用 learnw 函数产生一个标准的线性网络，其中使用了 learnwh 学习函数.

为了训练一个特定网络的第 i 层的权值和阈值使用 learnwh 函数来学习，需进行如下设置：

1）设置 NET.trainFcn 为 trainwb.（NET.trainParam 将自动设为 trainwb 的缺省值.）

2）设置 NET.adaptFcn 为 adaptwb.（NET.trainParam 将自动设为 trainwb 的缺省

值.）

3）设置第 i 层的 NET. inputWeights{i,j}. learnFcn 为一个 learnwh 学习函数. 同样，设置第 i 层的 NET. layerWeights{i,j}. learnFcn 为一个 learnwh 学习函数. 设置每一层的 NET. biases{i}. learnFcn 为一个 learnwh 学习函数.（网络的每个权值和阈值的学习参数将自动地被设为 trainwb 的缺省值.）

为训练一个网络（或使其自适应），需进行：

1）设置 NET. trainParam（NET. adaptParam）为希望值.

2）调用 train（或 adapt）函数.

自适应和训练的例子参见 newlin 函数.

算法：按照 Widrow-Hoff 学习准则，learnwh 函数从一个给定的神经元的输入 P、误差 E 以及权值（或阈值）学习率，可计算出该神经元的权值变化：

$$dw = lr * e * pn'$$

4. 分析函数

(1) maxlinlr　计算线性层的最大学习率.

格式：lr = maxlinlr(P)

　　lr = maxlinlr(P,'bias')

说明：maxlinlr 函数用于为 newlin 计算学习率. 通常学习率越大，网络训练所需的时间越少. 但是如果学习率太大，学习过程就不稳定. 该函数是用来计算 Widrow-Hoff 算法的线性神经元层的最大学习率.

maxlinlr(P)函数的参数为网络的给定输入 P 向量，可返回一个不带阈值的线性层所需的最大学习率，该层仅用 P 中的向量进行训练.

maxlinlr(P,'bias') 可返回一个带有阈值的线性层所需的最大学习率.

例 7.8　下面我们定义了 4 组 2 维输入向量：

$$P = [1\ 2\ -4\ 7;\ 0.1\ 3\ 10\ 6]$$

对具有阈值的线性层求出其最大学习率.

lr = maxlinlr(P,'bias')

上述语句执行后：

lr = 0.0067

(2) errsurf 计算单个神经元的误差曲面

格式：e = errsurf(p,t,wv,bv,f)

说明：errsurf 函数用于计算单个神经元的误差曲面. errsurf(p,t,wv,bv,f)的输入参数为：

p	Q 组单输入向量
t	Q 组单目标向量
wv	权值 W 的行向量.
bv	阈值 B 的行向量.
f	传递函数.

且在 wv 和 bv 上的返回误差矩阵.

例 7.9 图 7.6 是单个输入 purelin 神经元,其中输入和目标分别为

p = [−6.0 −6.1 −4.1 −4.0 +4.0 +4.1 +6.0 +6.1];
t = [+0.0 +0.0 +.97 +.99 +.01 +.03 +1.0 +1.0];

当权值从−1 变化到 1,阈值从−2.5 变化到 2.5 时,计算单个 purelin 神经元的误差曲面.

```
wv = -1:.1:1;
bv = -2.5:.25:2.5;
es = errsurf(p,t,wv,bv,' purelin');
```

用 plotes 画出误差表面:

```
plotes(wv,bv,es,[60 30])
```

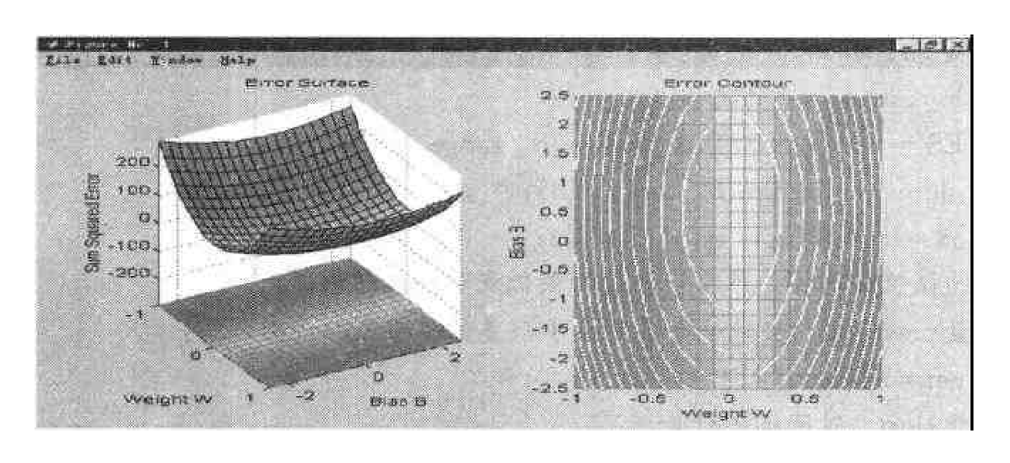


图 7.6 误差曲面

7.3 线性网络设计及应用举例

7.3.1 线性网络设计实例

1. 设计一个线性神经元

例 7.10 设计一个简单的单层线性神经元,使其实现从输入到输出的变换关系.其输入和目标分别为

$$P=[+1.0 \ -1.2]$$
$$T=[+0.5 \ +1.0]$$

该线性网络的结构如图 7.7 所示.

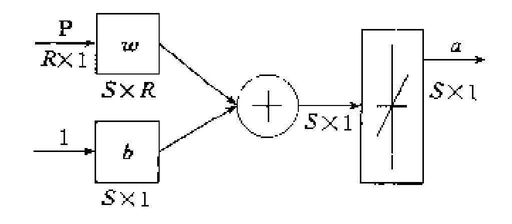


图7.7 线性网络结构图

给出权值和阈值的范围并绘制误差曲面及误差等高线，如图 7.8 所示.

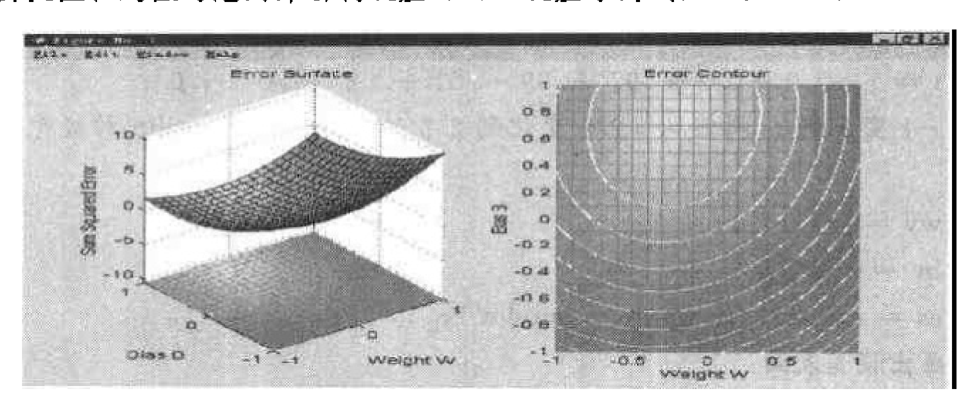


图 7.8　误差曲面及误差等高线

```
w_range = -1:0.1:1;
b_range = -1:0.1:1;
ES = errsurf(P,T,w_range,b_range,'purelin');
plotes(w_range,b_range,ES);
ax=findobj(gcf,'type','axes');
```

最优的权值和阈值是位于误差曲面最低点时的权值和阈值.

用函数 newlind( )设计一个具有最小误差的单层线性神经网络：

```
net=newlind(P,T);
```

用函数 sim( )对网络进行验证：

```
p=-1.2;
a=sim(net,p)
```

得到的结果为

```
a = 1
```

由此可见所设计的网络误差为零，该网络可以对输入精确求解.

下面给出本例的 MATLAB 源程序：

```
% dlin1.m
% NEWLIND     设计一个线性层
% SIM     仿真一个网络
clf;
figure(gcf);
P = [1.0 -1.2];
T = [0.5 1.0];
w_range = -1:0.1:1;
b_range = -1:0.1:1;
ES = errsurf(P,T,w_range,b_range,'purelin');
plotes(w_range,b_range,ES);
ax=findobj(gcf,'type','axes');
pause
```

```
net = newlind(P,T);
format compact;
A=0;E=0;SSE=0;
A = sim(net,P)
E = T - A
SSE = sumsqr(E)
plotes(w _ range,b _ range,ES);
plotep(net.IW{1,1},net.b{1},SSE);
p = -1.2;
a=0;
a = sim(net,p)
disp('End of dline1')
```

上述程序执行时可得到图 7.9 线性网络求解后的误差曲面及误差平方和等高线.

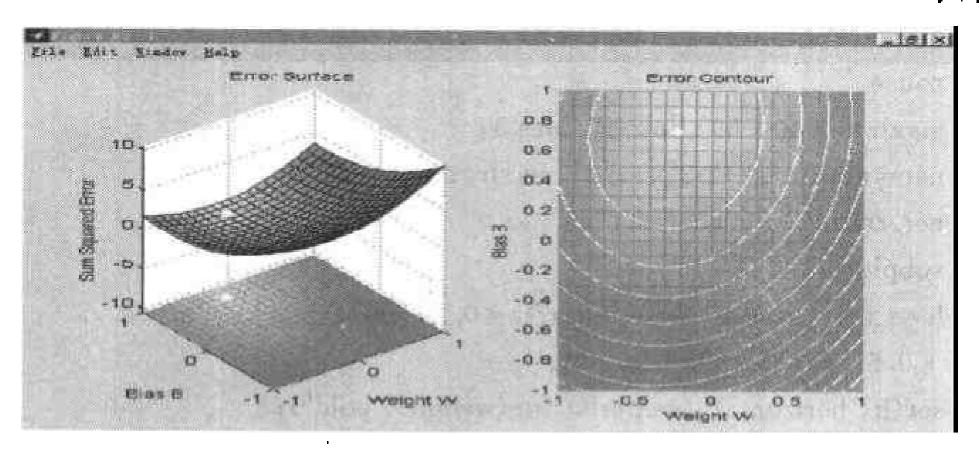


图 7.9　线性网络求解后的误差曲面及等高线

2. 对线性神经元进行训练

例 7.11　本例的问题与上例基本相同,只不过此处对线性网络进行训练以求最优解.

用 maxlinlr( )函数设置本例的学习率:

```
maxlr=0.4 * maxlinlr(P,'bias')
```

用 newlin( )建立一个线性网络.

```
net=newlin([-2 2],1,[0],maxlr);
```

设置训练的误差:

```
net.trainParam.goal=0.01;
```

由鼠标给出初始的权值和阈值,用函数 train( )训练线性网络:

```
[net,tr]=train(net,P,T);
```

用函数 sim( )对网络进行检验:

```
p=-1.2;
a=sim(net,p)
```

检验的结果为

```
a = 0.9888
```

下面给出本例的 MATLAB 源程序：

```
% dlin2
% NEWLIN      建立一个线性网络
% TRAIN      训练一个神经网络
% SIM      对神经网络进行仿真
clf;
figure(gcf)
P = [1.0 -1.2];
T = [0.5 1.0];
w_range = -1:0.2:1;
b_range = -1:0.2:1;
ES = errsurf(P,T,w_range,b_range,'purelin');
plotes(w_range,b_range,ES);
pause
maxlr=0.40 * maxlinlr(P,'bias');
net=newlin([-2 2],1,[0],maxlr);
net.trainParam.goal=.001;
subplot(1,2,2) ;
h = text(sum(get(gca,'xlim')) * 0.5,sum(get(gca,'ylim'))...
 * 0.5,' * Click On Me  * ');
set(h,'horizontal','center', 'fontweight','bold');
[net.IW{1,1} net.b{1}]=ginput(1);
delete(h);
[net,tr] = train(net,P,T);
format compact;
A=0;E=0;SSE=0;
A = sim(net,P)
E = T - A
SSE = sumsqr(E)
plotes(w_range,b_range,ES);
plotep(net.IW{1,1},net.b{1},SSE);
pause
plotperf(tr,net.trainParam.goal);
p = -1.2;
a = sim(net, p)
disp('End of dlin2')
```

上述程序执行后可得到如下结果：

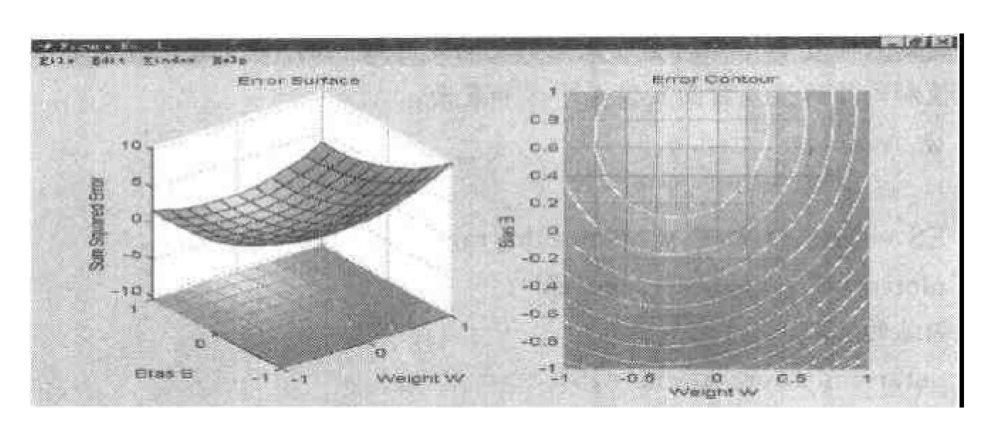


图 7.10　误差曲面及误差等高线

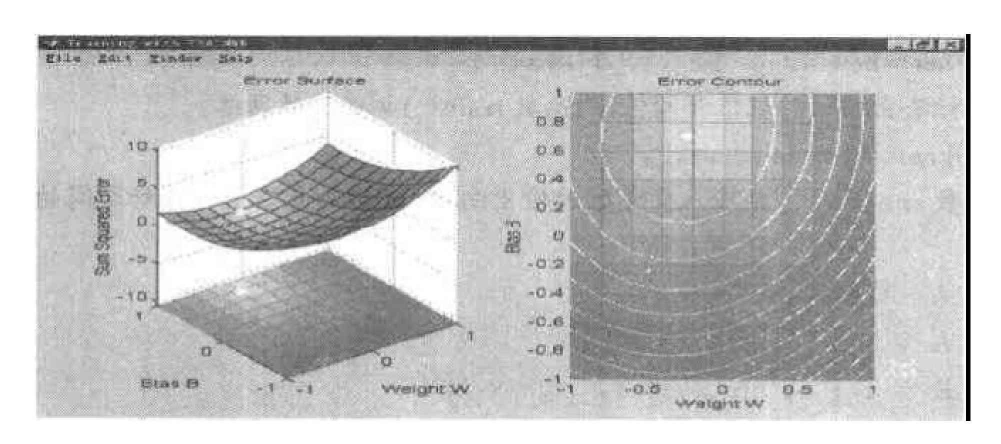


图 7.11　网络求解后的误差曲面及误差等高线

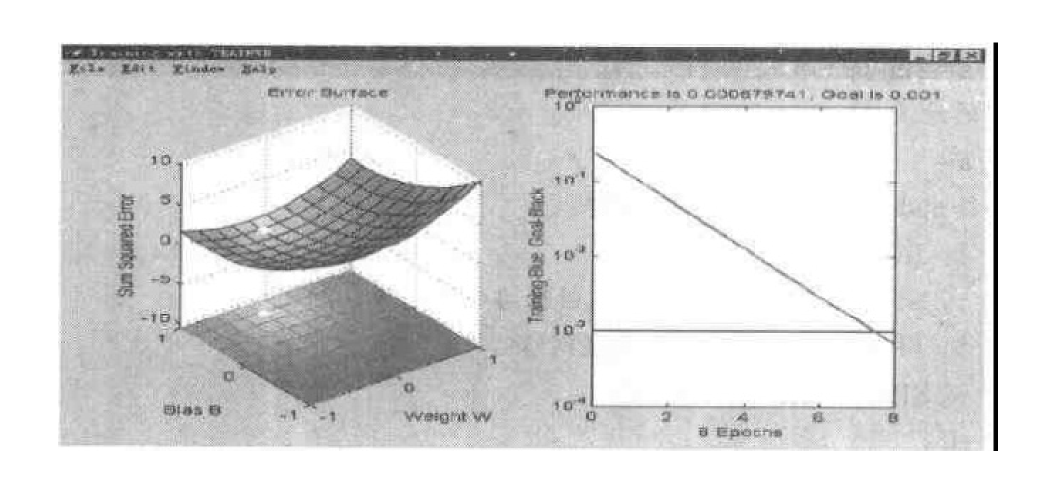


图 7.12　网络求解后误差曲面和训练参数与误差指标曲线

3. 非线性问题的线性适应

例 7.12　用一个线性网络对非线性问题进行线性适应，设输入样本向量和目标向量分别为

```
P = [+1.0 +1.5 +3.0 -1.2];
T = [+0.5 +1.1 +3.0 -1.0];
```

由上述输入向量和目标向量可见他们之间是非线性的. 首先给出权值和阈值的范围并绘制权值和阈值的误差曲面及误差平方和表面:

```
w_range =-2:0.4:2;
b_range = -2:0.4:2;
ES = errsurf(P,T,w_range,b_range,'purelin');
plotes(w_range,b_range,ES);
```

设置训练的次数:

```
net.trainParam.epochs=15;
```

用 maxlinlr( )函数设置本例的学习率:

```
maxlr=maxlinlr(P,'bias');
```

用函数 newlind( )设计一个具有最小误差的单层线性神经网络:

```
net=newlin([-2 2],1,[0],maxlr);
```

由鼠标给出初始的权值和阈值,用函数 train( )训练线性网络:

```
[net,tr]=train(net,P,T);
```

用函数 sim( )求给定输入向量时的输出向量,然后求出输出与目标之间的误差及均方误差:

```
A=0;E=0;SSE=0;
A = sim(net,P)
E = T - A
SSE = sumsqr(E)
```

用函数 sim( )对网络进行验证:

```
p=-1.2;
a=sim(net,p)
```

得到的结果为

```
a=-1.2487
```

下面给出本例的 MATLAB 源程序:

```
% dlin3
% NEWLIN    建立一个线性网络
% TRAIN    训练一个神经网络
% SIM    对网络进行仿真
clf;
figure(gcf)
P = [+1.0 +1.5 +3.0 -1.2];
T = [+0.5 +1.1 +3.0 -1.0];
w_range =-2:0.4:2;
b_range = -2:0.4:2;
ES = errsurf(P,T,w_range,b_r ange,'purelin');
```

```
plotes(w _ range,b _ range,ES);
net.trainParam.epochs=15;
maxlr=maxlinlr(P,'bias');
net=newlin([-2 2],1,[0],maxlr);
pause;
subplot(1,2,2) ;
    h = text(sum(get(gca,'xlim')) * 0.5,sum(get(gca,'ylim'))...
* 0.5,' * Click On Me *');
set(h,'horizontal','center', 'fontweight','bold');
[net.IW{1,1} net.b{1}]=ginput(1) ;
delete(h);
[net,tr] = train(net,P,T);
format compact;
A=0;E=0;SSE=0;
A = sim(net,P)
E = T - A
SSE = sumsqr(E)
plotes(w _ range,b _ range,ES);
plotep(net.IW{1,1},net.b{1},SSE);
pause;
plotperf(tr,net.trainParam.goal);
p = -1.2;
a = sim(net, p)
disp('End of dlin3')
```

下面给出该程序的仿真结果：

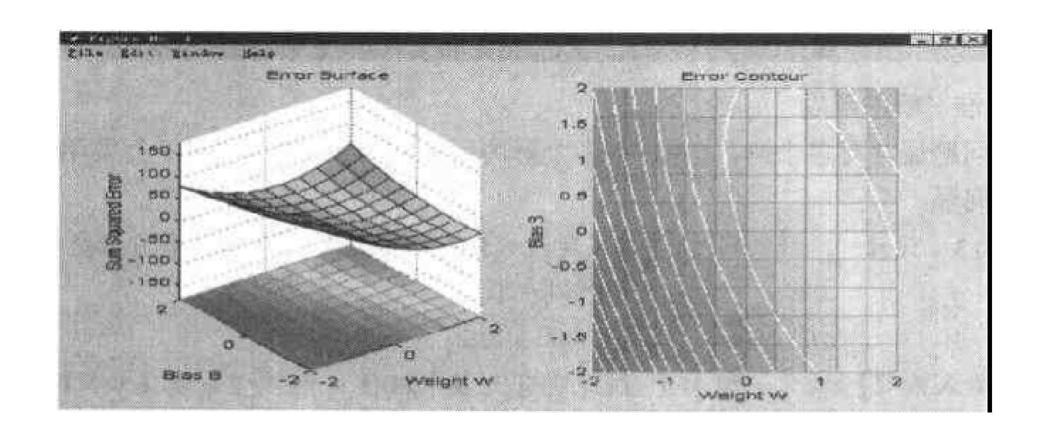


图 7.13 误差曲面及误差等高线

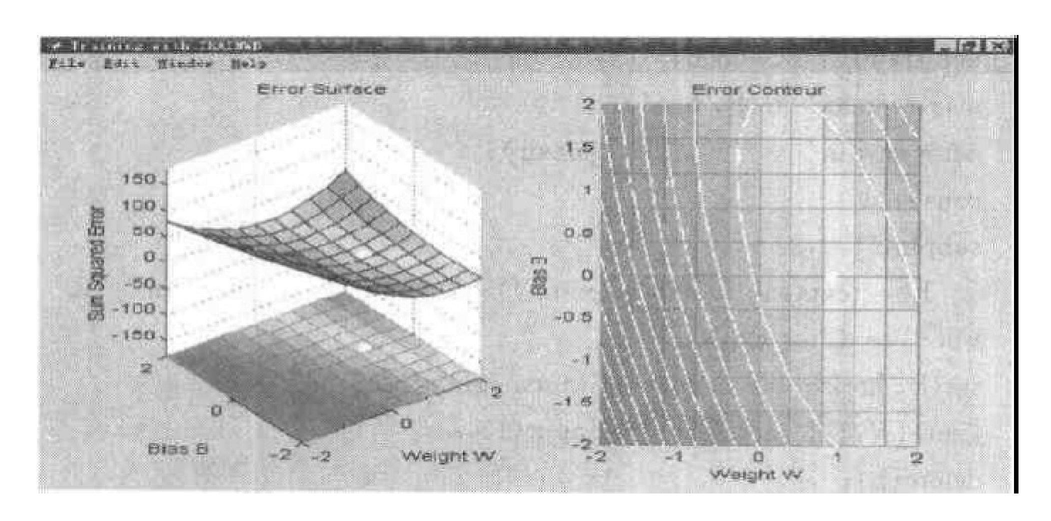


图 7.14 网络求解后的误差曲面及误差等高线

由图 7.15 可见误差不能达到零，这是因为这个问题是非线性的，用线性神经元模拟不可能得到零误差的解.

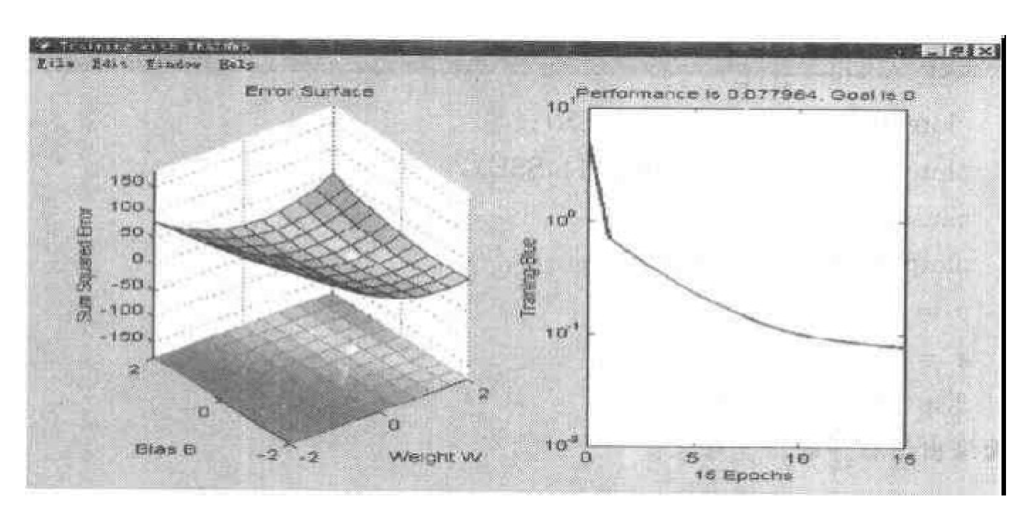


图 7.15 网络求解后的误差曲面及训练参数的误差指标曲线

4. 非惟一解的问题

有些问题通常有多个解，这种具有多个解的问题称为不确定问题. 下面采用线性网络求解这种问题.

例 7.13 设计一个线性网络，使其实现输入到目标的变换. 输入向量和目标向量分别为

$$P=[+1.0]$$

$$T=[+0.5]$$

由于输入和目标均为一维向量，因此将有无数个权值 w 和阈值 b 满足 $w\times P+b=T$，该问题是一个不确定问题.

给出权值和阈值的范围并绘制权值和阈值的误差曲面及误差等高线如图 7.16 所示.

```
w_range = -1:0.2:1;
b_range = -1:0.2:1;
```

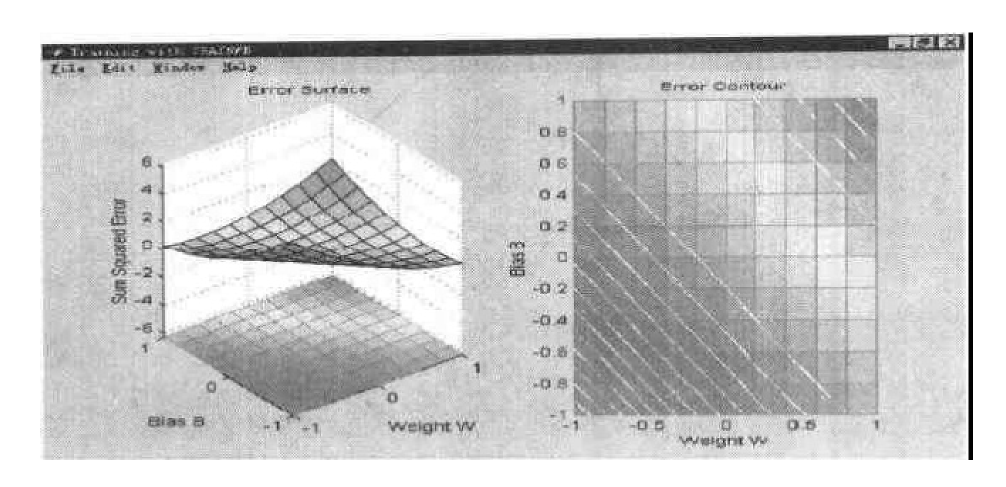


图 7.16　权值和阈值的误差曲面及误差等高线

ES = errsurf(P,T,w _ range,b _ range,'purelin');

plotes(w _ range,b _ range,ES);

用函数 newlind( )设计一个具有最小误差的单层线性神经网络:

net=newlind(P,T);

由鼠标给出初始的权值和阈值,用函数 train( )训练线性网络:

[net,tr]=train(net,P,T);

用函数 sim( )求给定输入向量时的输出向量,然后求出输出与目标之间的误差及均方误差:

A=0;E=0;SSE=0;

A = sim(net,P)

E = T - A

SSE = sumsqr(E)

绘制相应的权值和阈值的误差曲面及误差平方和表面以及训练参数的误差指标曲线.

用 train( )函数和 newlin( )函数求得的解是不一样的. train( )函数根据不同的初始的条件(权值和阈值)得到不同的解,而 newlin( )函数将总可得到相同的解.

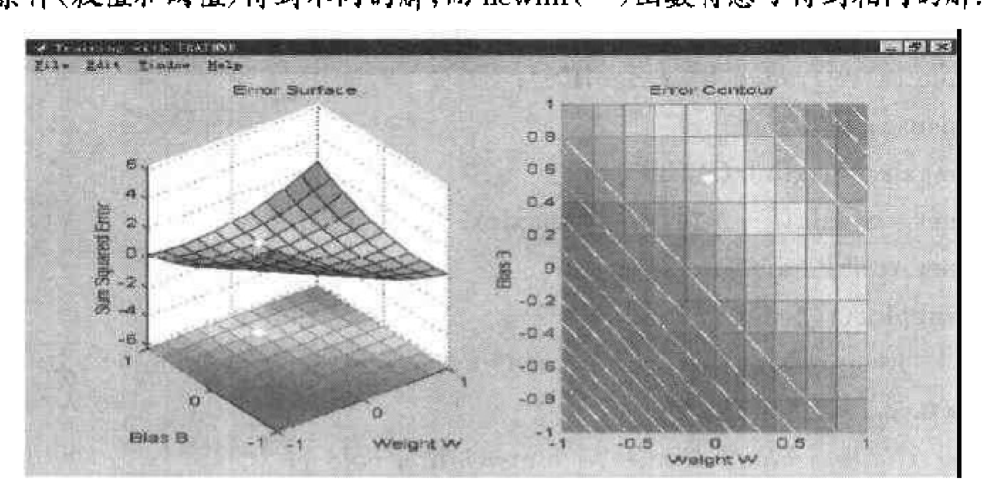


图 7.17　网络求解后的误差曲面及误差等高线

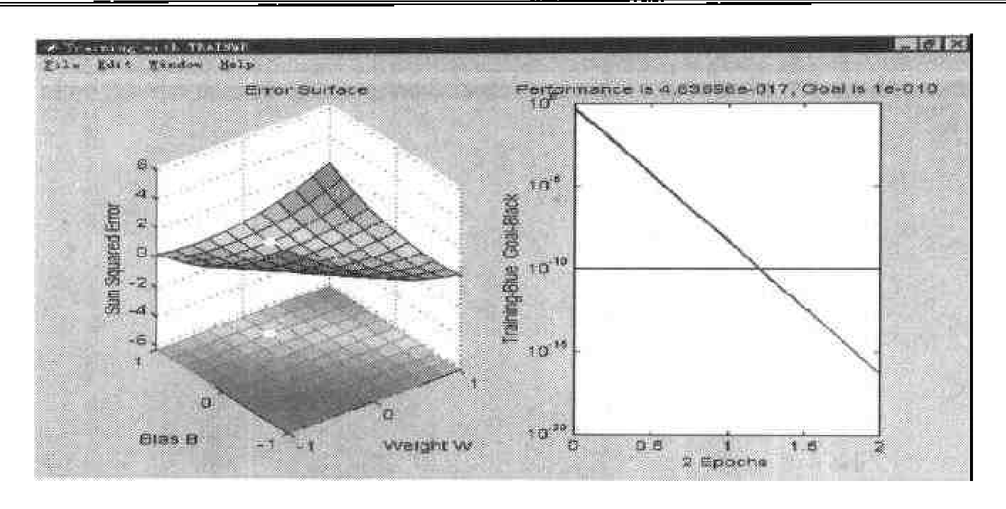


图 7.18　网络求解后的误差曲面和训练参数与误差指标曲线

用函数 sim(　)对网络进行验证：

```
p=+1.0;
a=sim(net,p)
```

得到的结果为

```
a = 0.5000
```

下面给出本例的 MATLAB 源程序：

```
% dlin4
% NEWLIN    建立一个线性网络
% TRAIN    训练一个神经网络
% SIM    对神经网络进行仿真
clf;
figure(gcf)
P = [+1.0];
T = [+0.5];
w_range = -1:0.2:1; b_range = -1:0.2:1;
ES = errsurf(P,T,w_range,b_range,'purelin');
plotes(w_range,b_range,ES);
pause;
maxlr=maxlinlr(P,'bias');
net=newlin([-2 2],1,[0],maxlr);
net.trainParam.goal=1e-10;
subplot(1,2,2) ;
    h=text(sum(get(gca,'xlim'))*0.5,sum(get(gca,'ylim')) ...
*0.5,'* Click On Me *');
set(h,'horizontal','center', 'fontweight','bold');
[net.IW{1,1} net.b{1}]=ginput(1) ;
delete(h);
```

```
[net,tr] = train(net,P,T);
format compact;
A=0;E=0;SSE=0;
A = sim(net,P)
E = T - A
SSE = sumsqr(E)
plotes(w _ range,b _ range,ES);
plotep(net.IW{1,1},net.b{1},SSE);
pause;
subplot(1,2,2) ;
plotperf(tr,net.trainParam.goal);
pause;
plotes(w _ range,b _ range,ES);
plotep(net.IW{1,1},net.b{1},SSE);
net = newlind(P,T);
hold on;
plot(net.IW{1,1},net.b{1},'ro')
hold off;
p = 1.0;
a = sim(net,p)
disp('End of dlin4')
```

5. 线性相关问题

对于输入向量之间线性相关的问题,用线性网络求解.如果线性相关的输入向量与输出之间并不匹配,则此问题是一个非线性问题,且这一问题得不到零误差解.

例 7.14　对于下列输入样本向量和目标向量

```
P = [1.0  2.0  3.0; 4.0  5.0  6.0];
T = [0.5 1.0 -1.0];
```

用线性网络设计此问题.

由上述给出的输入向量和目标向量可见这是一个输入线性相关的问题,而且输入与输出之间不匹配.

用 newlin 函数建立一个线性网络:

```
net=newlin([0 10;0 10],1,[0],maxlr);
```

设置如下训练参数:

```
net.trainParam.show = 50;
net.trainParam.epochs =500;
net.trainParam.goal = 0.001;
```

用 train 训练网络:

```
[net,tr]=train(net,P,T);
```

采用一个输入向量验证训练后的网络:

```
p = [1.0; 4];
a = sim(net,p)
```

仿真的结果为

```
a = 0.8971
```

这一结果不是 0.5,因此线性网络不能适应一个输入之间是线性相关的非线性问题.

下面给出本例的 MATLAB 源程序:

```
% dlin5
% NEWLIN      建立一个线性网络.
% TRAIN       训练一个神经网络.
% SIM         仿真一个神经网络.
clf;
figure(gcf)
P = [1.0  2.0  3.0; 4.0  5.0  6.0];
T = [0.5 1.0 -1.0];
maxlr=maxlinlr(P,'bias');
net=newlin([0 10;0 10],1,[0],maxlr);
net.trainParam.show = 50;
net.trainParam.epochs =500;
net.trainParam.goal = 0.001;
more off;
[net,tr]=train(net,P,T);
set(gcf,'name','Linearly Dependent Problem');
p = [1.0; 4];
a = sim(net,p)
disp('End of dlin5')
```

下面给出该程序仿真的结果如图 7.19 所示.

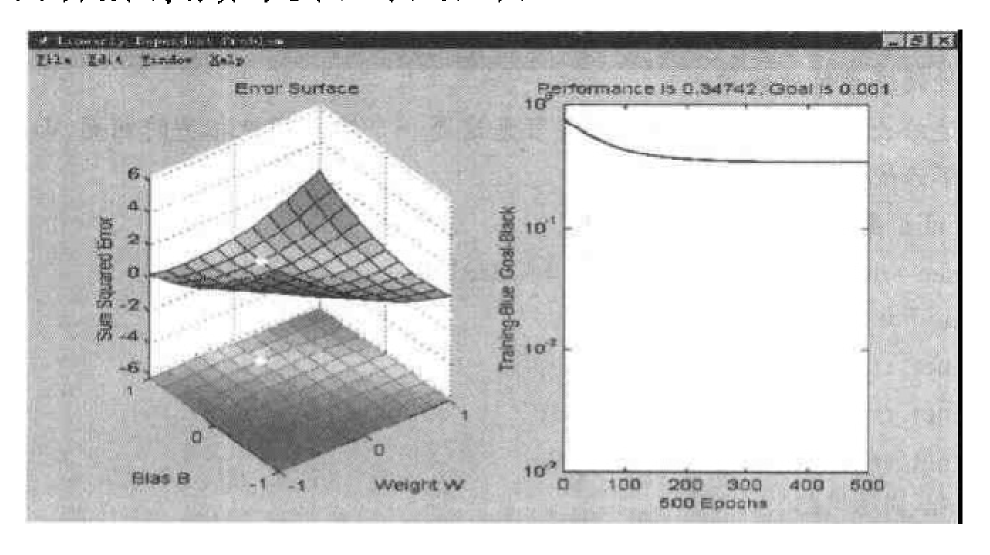


图 7.19　网络求解后的误差曲面及训练参数的误差曲线

由仿真结果可见该问题不能达到要求的误差指标. 这是因为输入向量之间是线性相关的.

6. 学习速率问题

在网络设计中,学习速率的选取是影响收敛速度以及训练结果的一个很重要的因素. 只要学习速率足够小,采用 Widrow-Hoff 学习规则的线性网络总可以训练出一个最小的输出误差. 但是当学习速率较大时,可导致训练过程的不稳定. MATLAB 工具箱给出了一个正确求解学习率的函数 maxlinlr. 下面我们用比 maxlinlr 求得的学习率大的值训练网络.

例 7.15 输入和目标与例 7.10 相同,但学习率是原来的 1.5 倍.

本例选取的学习率为

```
maxlr=maxlinlr(P,'bias');
```

训练次数为

```
net.trainParam.epochs=20;
```

下面给出本例的 MATLAB 源程序:

```
% dlin6
% NEWLIN    建立一个线性网络.
% TRAIN    训练一个神经网络.
% SIM    仿真一个神经网络.
clf;
figure(gcf)
P = [+1.0 -1.2];
T = [+0.5 +1.0];
w_range = -2:0.4:2;
b_range = -2:0.4:2;
ES = errsurf(P,T,w_range,b_range,'purelin');
plotes(w_range,b_range,ES);
maxlr=maxlinlr(P,'bias');
net=newlin([-2 2],1,[0],maxlr * 2.25) ;
net.trainParam.epochs=20;
pause;
subplot(1,2,2) ;
    h = text(sum(get(gca,'xlim')) * 0.5,sum(get(gca,'ylim'))...
* 0.5,' * Click On Me  * ');
set(h,'horizontal','center', 'fontweight','bold');
[net.IW{1,1} net.b{1}]=ginput(1) ;
delete(h);
[net,tr]=train(net,P,T);
plotperf(tr,net.trainParam.goal);
p = -1.2;
```

```
a = sim(net, p)
disp('End of dlin6')
```

该程序的仿真结果为

```
a = 66.2114
```

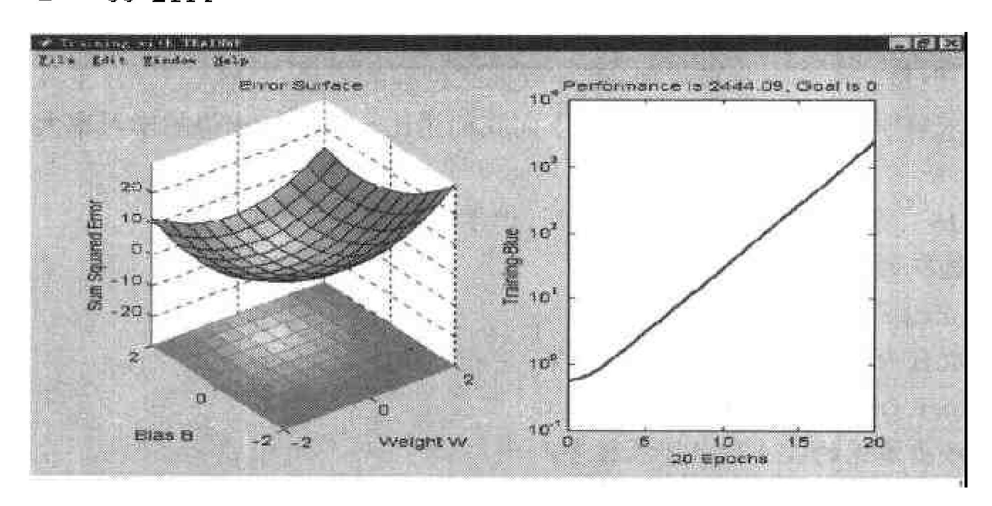


图 7.20 网络求解后的误差曲面及训练参数的误差曲线

由仿真结果可见，当学习率选取较大时，误差越来越大，网络得不到一个最小误差解. 因此学习率的选取是非常关键的，应选取较小的学习率以保证网络的收敛，而不应选太大的学习率.

7. 设计自适应线性网络

例 7.16 设计一个自适应线性网络，使其对输入信号进行预测. 输入信号 P 和目标信号 T 为

```
time = 1:0.01:2.5;
X = sin(sin(time).*time*10) ;
P = con2seq(X);
T = con2seq(2*[0 X(1:(end-1))] + X);
```

首先绘出上述信号的图形，如图 7.21 所示.

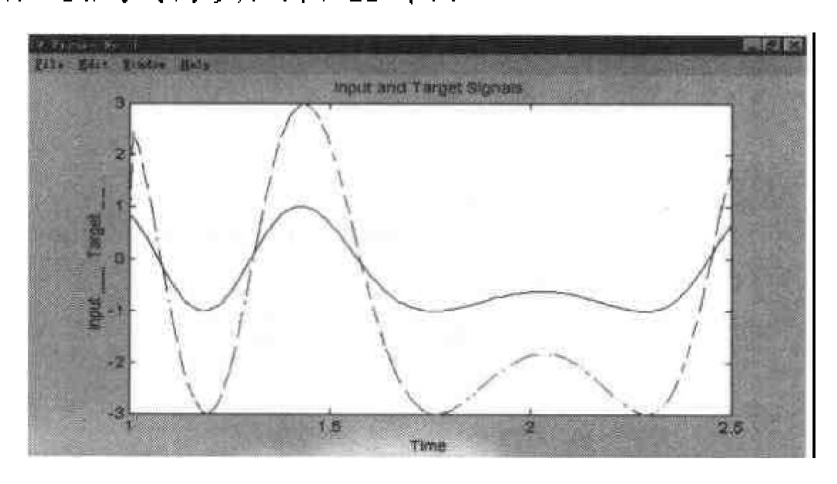


图 7.21 输入和目标信号

下面给出本例的 MATLAB 源程序：

```
% dlin7
% NEWLIN Creates a linear layer.',
% TRAIN Trains a neural network.',
% ADAPT Allows a neural network to adapt.',
time = 1:0.01:2.5;
X = sin(sin(time).* time * 10) ;
P = con2seq(X);
T = con2seq(2 * [0 X(1:(end-1) )] + X);
plot(time,cat(2,P{:}),time,cat(2,T{:}),'--')
title('Input and Target Signals')
xlabel('Time')
ylabel('Input \_\_\_ Target \_ \_')
pause
net = newlin([-3 3],1,[0 1],0.1) ;
plot(time,cat(2,T{:}),'--');
hold on;
plot([1 2.5],[0 0],'k');
title('Output and Target and Error Signals');
xlabel('Time');
ylabel('Output \_\_\_ Target \_ \_ Error \_ .');
pause
YN=0; EN=0; TimeN=time(1) ;
for i=2:
length(P);
TimeO=TimeN;
TimeN=time(i);
YO=YN;
EO=EN;
[net,Y,E,Pf]=adapt(net,P{i},T{i});
YN=Y(1) ;
EN=E(1) ;
plot([TimeO TimeN],[YO YN],'-',[TimeO TimeN],[EO EN],'-.');
drawnow;
end;
hold off;
disp('End of dlin7')
```

仿真结果如图 7.22 所示.

由仿真结果可见在 2 秒时输出与目标之间的误差接近于零,因此该线性网络可以对输入进行预测.

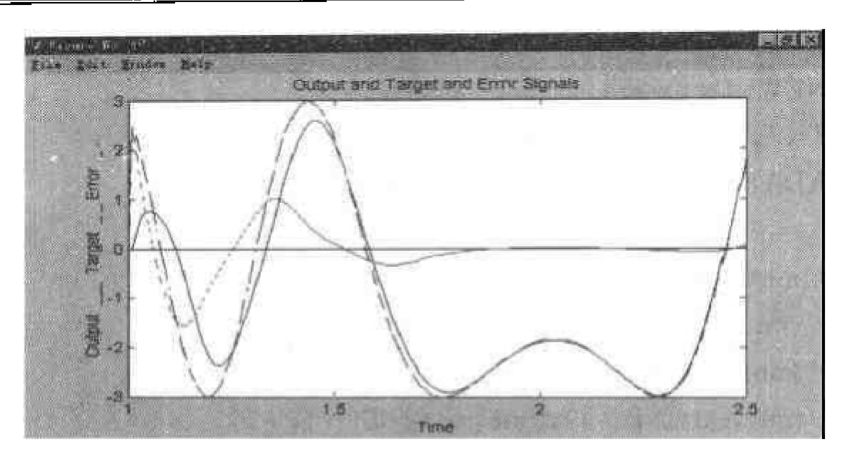


图 7.22　输出、目标及误差曲线

7.3.2　线性网络应用实例

1. 利用线性网络进行预测

例 7.17　用线性网络预测时间序列的未来值. 在已知信号的 5 个 过去值的情况下预测将来值.

1）问题描述

信号 T 的特征为：持续 5 秒，采样频率为每秒 40 次. 则信号 T 为

```
time = 0:0.025:5;
T = sin(time * 4 * pi);
```

网络输入 P 是将信号 T 分别延迟 1 至 5 个时间单位得到的 5 个分量.

```
P = zeros(5,Q);
P(1,2:Q) = T(1,1:(Q-1) );
P(2,3:Q) = T(1,1:(Q-2) );
P(3,4:Q) = T(1,1:(Q-3) );
P(4,5:Q) = T(1,1:(Q-4) );
P(5,6:Q) = T(1,1:(Q-5) );
```

图 7.23 即是要用网络来预测的函数的信号 T.

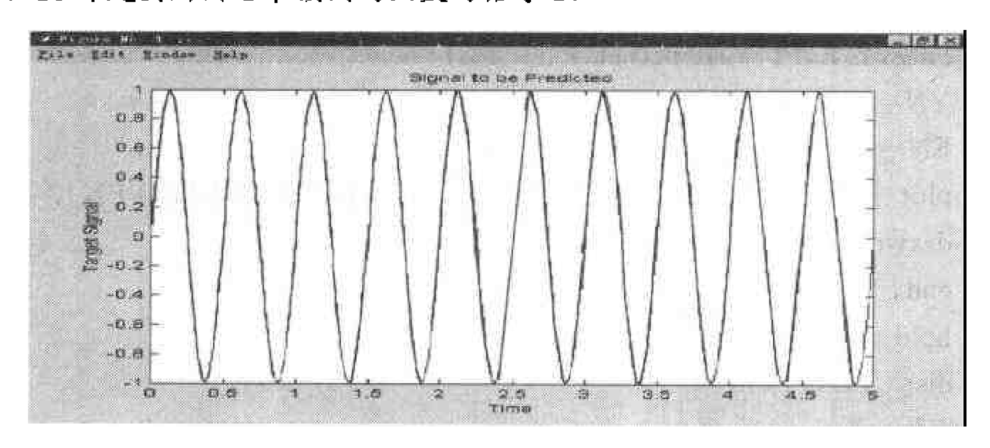


图 7.23　信号 T

2）网络设计

由于网络有 5 个输入信号和一个输出信号(预测值)，故采用有 5 个输入的单个线性神经元网络. 网络结构如图 7.24 所示.

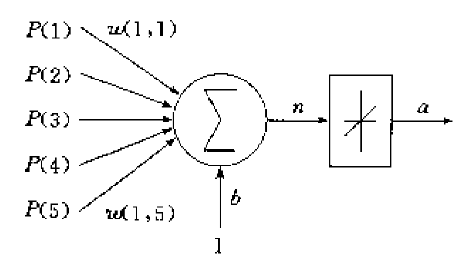


图7.24　网络结构

利用 newlind 函数来设计这个网络，即可得网络的权值和阈值.

3）网络测试

对设计好的网络测试其预测性能. 用上述 5 个延迟的信号作为样本输入，利用 sim 函数求出网络的输出 a，然后将 a 与信号的实际值 T 作比较，结果如图 7.25 所示，同时也可得到预测误差，如图 7.26 所示. 从误差曲线可见预测信号与实际信号相当接近.

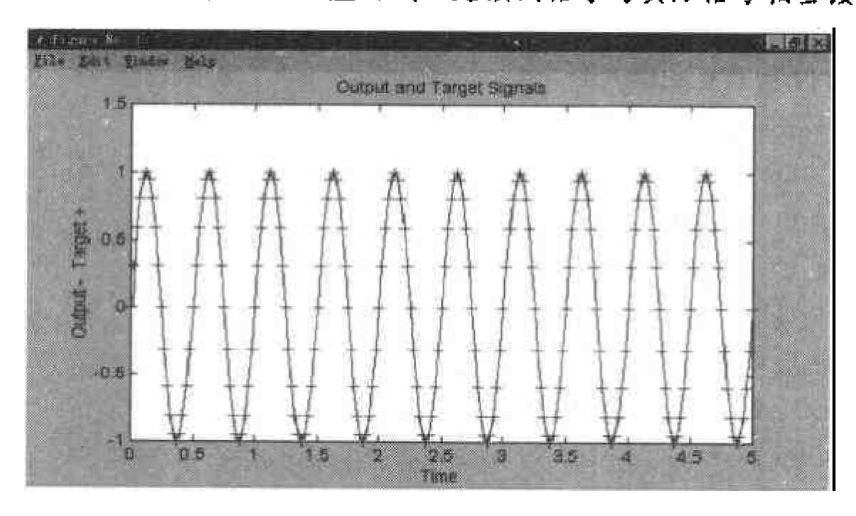


图 7.25　信号 T 与网络预测输出

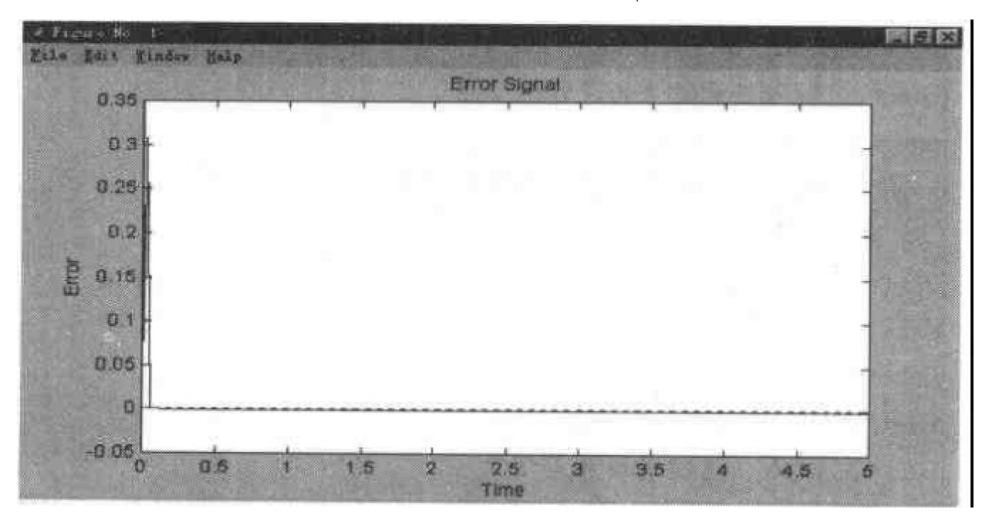


图 7.26　预测误差

4) 从上面的误差曲线可见，在预测的初始阶段，误差较大，但经过一段时间之后，误差几乎趋于零. 这是由于网络需要 5 个延迟的信号作为输入，而在初始阶段，网络的输入不完整，因此，不可避免出现初始误差. 由结果可见，线性网络对于线性时间序列的预测问题是有实用价值的.

5) 线性预测的 MATLAB 程序

```
% APPLIN1
clf;
figure(gcf)
echo on
%  NEWLIND     设计一个线性层.
%  SIM     对线性层进行仿真.
pause % 键入任意键继续
% 定义一个信号波形
time = 0:0.025:5; % 从 0 到 6 秒
T = sin(time * 4 * pi);
Q = length(T);
% 由信号 T 生成输入 P
P = zeros(5,Q);
P(1,2:Q) = T(1,1:(Q-1) );
P(2,3:Q) = T(1,1:(Q-2) );
P(3,4:Q) = T(1,1:(Q-3) );
P(4,5:Q) = T(1,1:(Q-4) );
P(5,6:Q) = T(1,1:(Q-5) );
pause % 键入任意键可以观察信号波形
plot(time,T)
xlabel('Time');
ylabel('Target Signal');
title('Signal to be Predicted');
pause % 键入任意键开始设计网络
net = newlind(P,T);
pause % 键入任意键开始测试网络的性能
a = sim(net,P);
plot(time,a,time,T,'+')
xlabel('Time');
ylabel('Output - Target +');
title('Output and Target Signals');
pause % 键入任意键观察误差信号
e = T-a;
plot(time,e)
```

```
hold on
plot([min(time) max(time)],[0 0],':r')
hold off
xlabel('Time');
ylabel('Error');
title('Error Signal');
echo off
disp('End of APPLIN1')
```

2. 利用线性神经网络进行自适应预测

例 7.18　利用函数 adapt 对线性网络进行自适应训练，在线修正网络的权值和阈值，这样对于时变信号，网络就可以及时跟踪其变化，即可对时变信号序列进行预测.

1) 问题描述

待预测的信号 T 的特征为：信号持续时间为 6 秒，前 4 秒每秒采样 20 次，从第 4 秒开始，频率加倍，即 4 至 6 秒的信号频率是 0 至 4 秒信号频率的两倍. 定义信号 T 为

```
time1 = 0:0.05:4;
time2 = 4.05:0.024:6;
time = [time1 time2];
T = con2seq([sin(time1 * 4 * pi) sin(time2 * 8 * pi)]);
```

输入信号如图 7.3 所示.

在每个采样时间点，该时刻之前的 5 个采样信号值被选定作为网络的输入，网络的输出就是下一时刻信号的预测值.

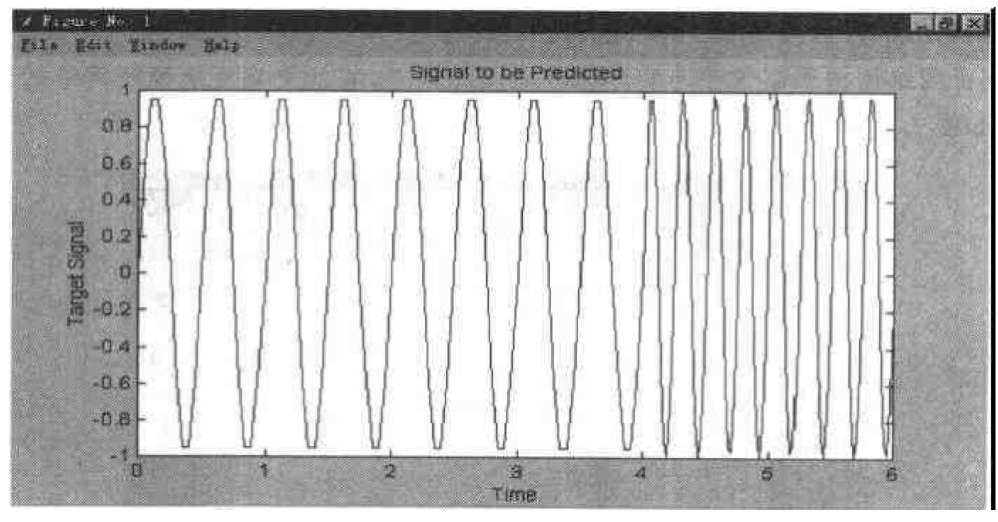


图 7.27　输入信号

2) 生成一个线性网络

采用具有 5 个反馈输入和 1 个输出的单个线性神经元的网络，该网络的输出值即为下一时刻的信号预测值，线性网络的学习率为 0.1，使用函数 newlin 生成该网络：

```
lr = 0.1;
delays = [1 2 3 4 5];
net = newlin(minmax(cat(2,P{:})),1,delays,lr);
```

3）线性网络的训练

使用 atapt 函数对线性网络进行权值和阈值的自适应修正. 该函数可对初始给定的线性网络，在已知的输入信号和目标信号下，对信号进行滤波.

```
[net,y,e]=adapt(net,P,T);
```

4）网络性能测试

对训练后的网络，将预测信号和实际信号绘制在同一幅图中对照比较，如图 7.28 所示.

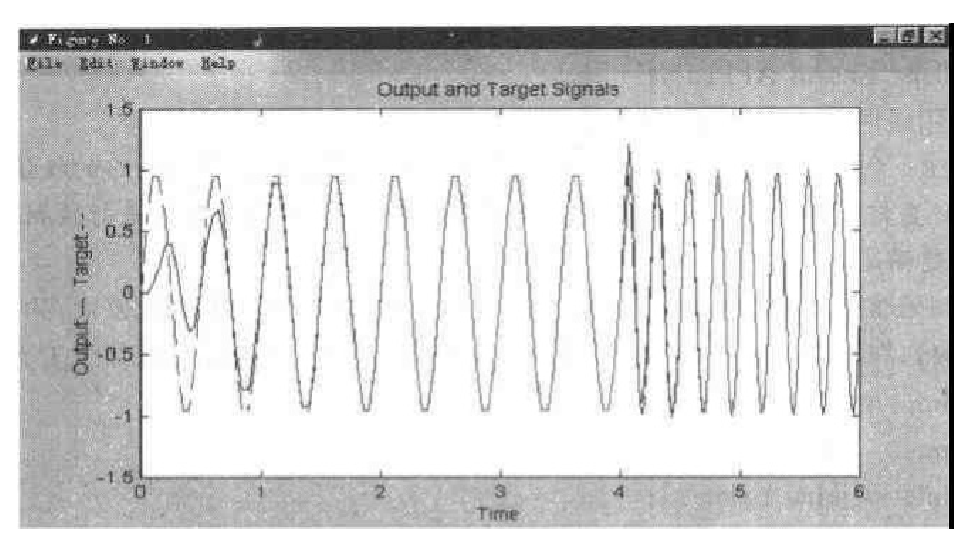


图 7.28　预测信号和实际信号

从图中可见，在最初的 1.5 秒时间内，网络在跟踪目标信号；之后，网络即可以准确地预测目标信号，两条曲线几乎完全重合；在第 4 秒时，信号的频率发生突变，网络输出的预测信号曲线和实际的目标信号曲线稍有偏差，但因为网络能进行自适应学习，在很短的时间之后，网络的输出又能比较精确地预测目标信号.

绘制出网络输出预测信号与实际信号之间的误差，将能更直观地说明网络的预测性能，误差如图 7.29 所示.

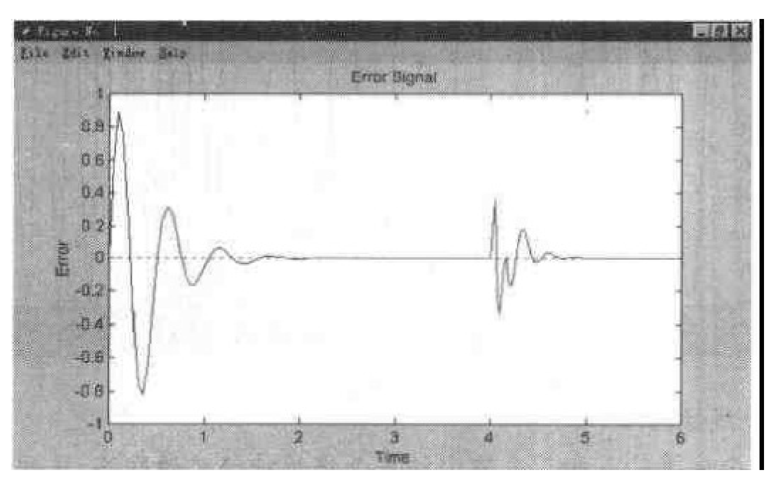


图 7.29　预测误差

5）自适应预测的 MATLAB 程序

```
%APPLIN2
clf;
```

```
figure(gcf)
echo on
%      NEWLIN      生成一个初始化的线性神经网络
%      ADAPT       用 Widrow-Hoff 学习规则训练线性网络
pause % 键入任意键继续
%      用 TIME1 及 TIME2 定义两段时间
time1 = 0:0.05:4;       % from 0 to 4 seconds
time2 = 4.05:0.024:6; % from 4 to 6 seconds
%      TIME 是整个仿真时间
time = [time1 time2]; % from 0 to 6 seconds
T = con2seq([sin(time1 * 4 * pi) sin(time2 * 8 * pi)]);
P = T;
pause % 键入任意键可观察目标信号
plot(time,cat(2,T{:}))
xlabel('Time');
ylabel('Target Signal');
title('Signal to be Predicted');
pause % 键入任意键开始设计网络
lr = 0.1;
delays = [1 2 3 4 5];
net = newlin(minmax(cat(2,P{:})),1,delays,lr);
pause % 键入任意键开始对网络进行自适应训练
[net,y,e]=adapt(net,P,T);
pause % 键入任意键观察输出波形
plot(time,cat(2,y{:}),time,cat(2,T{:}),'--')
xlabel('Time');
ylabel('Output --- Target - -');
title('Output and Target Signals');
pause % 键入任意键观察误差信号波形
plot(time,cat(2,e{:}),[min(time) max(time)],[0 0],':r')
xlabel('Time');
ylabel('Error');
title('Error Signal');
echo off
disp('End of APPLIN2')
```

3. 线性系统辨识

线性网络可以用于对实际系统建模. 如果实际系统是线性的或是接近线性的，用线性网络建模的模型误差将非常小.

例 7.19　用线性网络对实际系统建模.

1）问题描述

对于有限冲击响应线性系统，有输入信号 X，其周期为 5s，每 25ms 采样 1 次。信号 X 为

```
time = 0:0.025:5;
X = sin(sin(time).*time*10);
```

该信号的图形如图 7.30 所示。

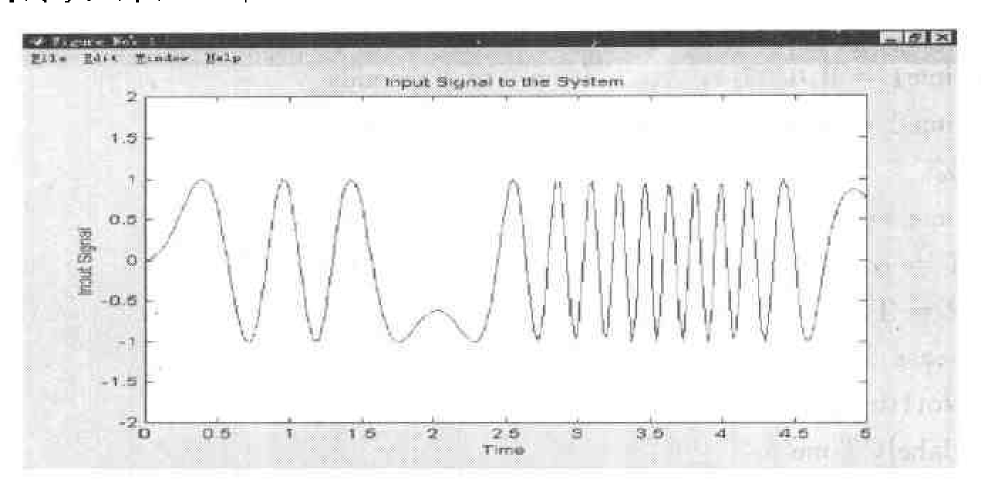


图 7.30　信号 X

网络的输入是输入信号 X 的当前值和前两个时刻的值，可由下列语句得到

```
Q=size(X,2) ;
P = zeros(3,Q);
P(1,1:Q) = X(1,1:Q);
P(2,2:Q) = X(1,1:(Q-1) );
P(3,3:Q) = X(1,1:(Q-2) );
```

假设系统输出可测，那么系统输出的测量值可由下列语句求得

```
T = filter([1 0.5 -1.5],1,X);
```

2）网络的设计

网络具有 3 个输入和 1 个输出，用下列语句产生该线性网络：

```
net = newlind(P,T);
```

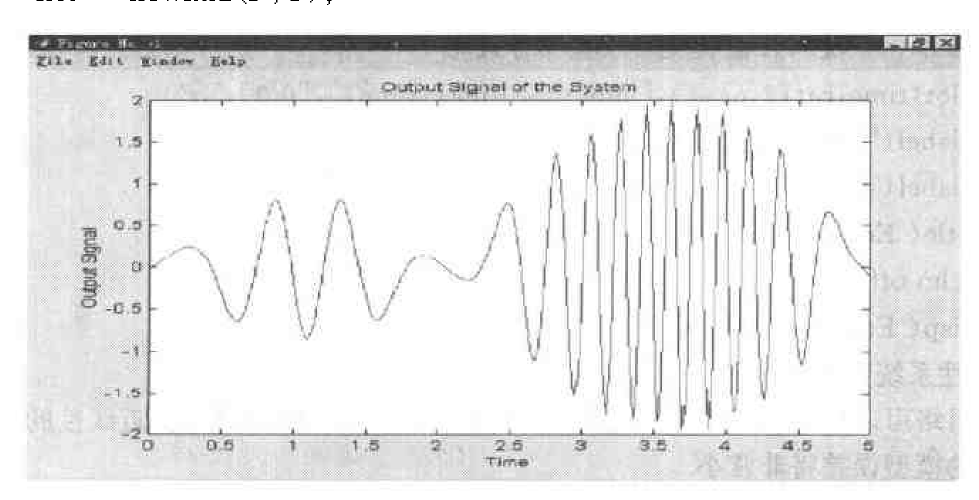


图 7.31　系统输出

3）网络性能测试

网络的权值和阈值一经确定，就可以利用函数 sim()对其进行性能测试：

```
a = sim(net,P);
```

将网络输出 a 和系统输出 t 的图形绘制在一张图中，如图 7.32 所示.

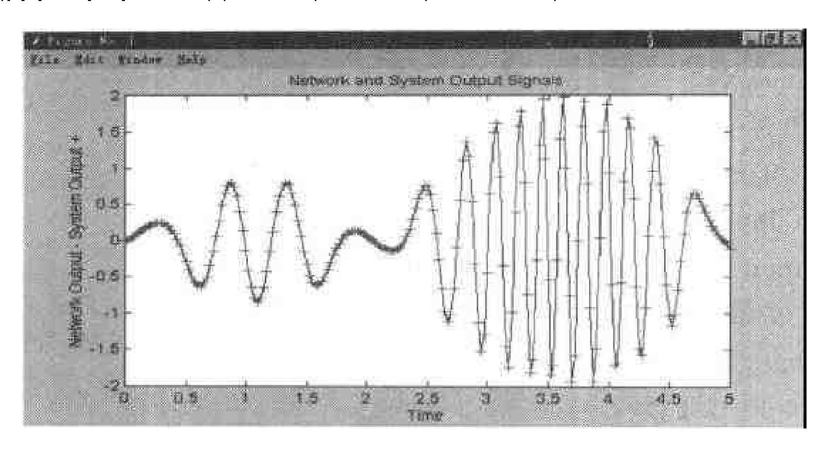


图 7.32　网络输出 a 和系统输出 t

图 7.33 给出了系统输出与网络输出的误差曲线，由此可见线性网络可以对系统进行精确建模.

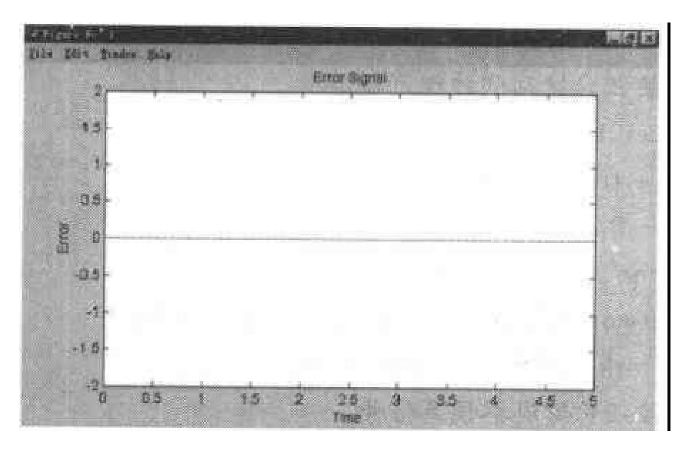


图 7.33　误差曲线

4）线性系统辨识的 MATLAB 程序

```
%APPLIN3
clf;
figure(gcf)
echo on
%    newlind    设计一个线性网络
%    SIMULIN    对线性网络进行仿真
pause % 按任意键继续
time = 0:0.025:5; % from 0 to 6 seconds
X = sin(sin(time).*time*10) ;
Q=size(X,2) ;
P = zeros(3,Q);
```

```
P(1,1:Q) = X(1,1:Q);
P(2,2:Q) = X(1,1:(Q-1));
P(3,3:Q) = X(1,1:(Q-2));
T = filter([1 0.5 -1.5],1,X);
pause % 按任意键观察信号 X 的图形
plot(time,X)
axis([0 5 -2 2]);
xlabel('Time');
ylabel('Input Signal');
title('Input Signal to the System');
pause % 按任意键察看系统输出波形
plot(time,T)
axis([0 5 -2 2]);
xlabel('Time');
ylabel('Output Signal');
title('Output Signal of the System');
pause % 按任意键开始设计网络
net = newlind(P,T);
pause % 按任意键测试神经元模型
a = sim(net,P);
plot(time,a,time,T,'+')
axis([0 5 -2 2]);
xlabel('Time');
ylabel('Network Output - System Output +');
title('Network and System Output Signals');
pause % 按任意键观察误差信号
e = T-a;
plot(time,e,[min(time) max(time)],[0 0],':r')
axis([0 5 -2 2]);
xlabel('Time');
ylabel('Error');
title('Error Signal');
echo off
disp('End of APPLIN3')
```

4. 自适应系统辨识

例 7.20 用线性网络对线性系统进行自适应系统辨识，当被辨识的线性系统发生变化时，线性网络通过自适应训练，实现对线性系统进行自适应辨识.

1) 问题描述

输入信号 X 的特征为：持续 6s 时间，且每分钟采样 200 次. 该输入信号为

```
time1 = 0:0.005:4;          % from 0 to 4 seconds
time2 = 4.005:0.005:6;      % from 4 to 6 seconds
time = [time1 time2];       % from 0 to 6 seconds
X = sin(sin(time * 4) .* time * 8) ;
```

信号 X 的波形如图 7.34 所示.

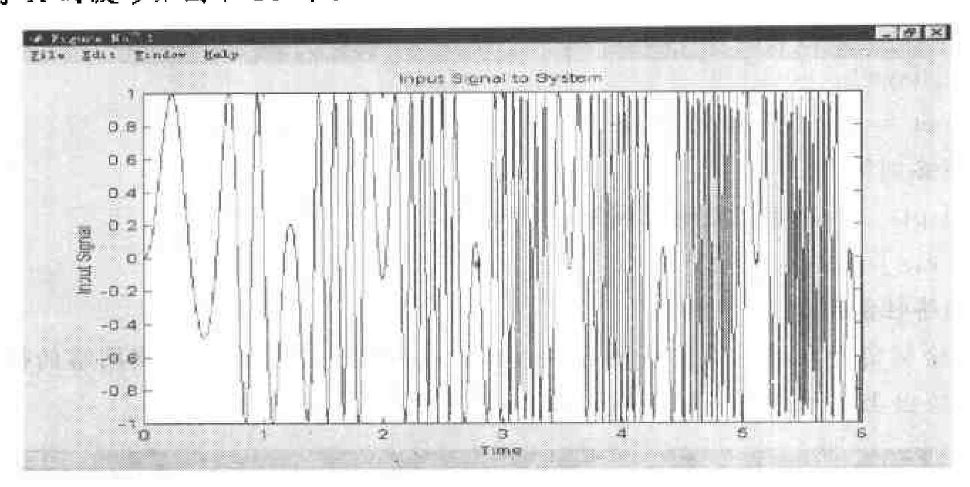


图 7.34 输入信号 X

系统的输出信号定义为

```
steps1 = length(time1) ;
[T1,state] = filter([1 -0.5],1,X(1:steps1) );
steps2 = length(time2) ;
T2 = filter([0.9 -0.6],1,X((1:steps2) + steps1) ,state);
T = [T1 T2];
```

图 7.35 给出系统输出信号的图形. 在 4s 时刻之前的信号和之后的信号之间的区别不太明显,但是有区别的.

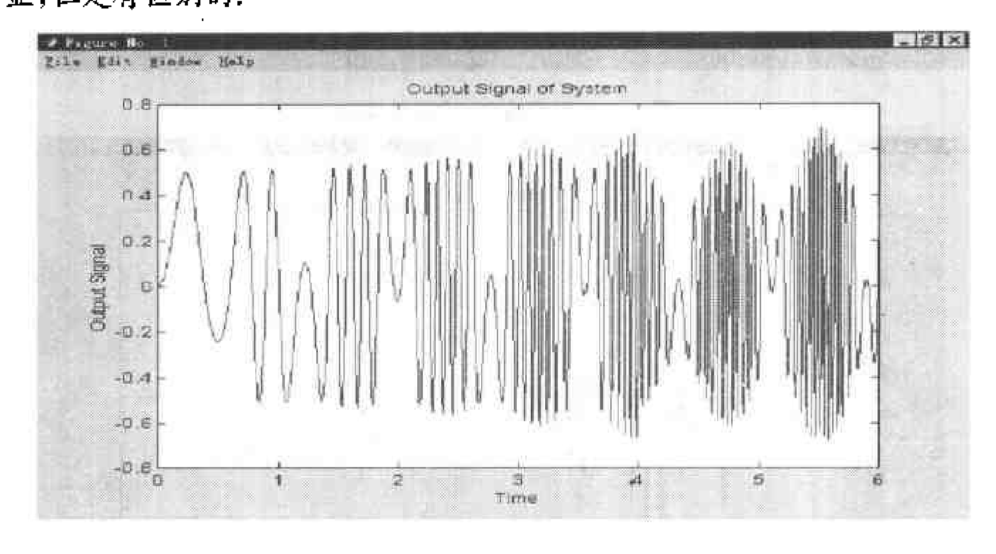


图 7.35 系统输出

用输入信号 X 当前时刻的值和前一采样时刻的值作为线性网络的 P.网络输入可用下列语句得到

```
T = con2seq(T);
P = con2seq(X);
```

2) 网络设计

用函数 newlin( )建立一个线性网络. 网络具有 2 个延迟输入,设学习率为 0.5 ,可用下列语句实现:

```
lr = 0.5;
delays = [0 1];
net = newlin(minmax(cat(2,P{:})),1,delays,lr);
```

3) 网络训练

用 adapt( )函数对上述线性网络进行训练:

```
[net,y,e]=adapt(net,P,T);
```

4) 网络性能测试

通过绘制实际系统的输出 t 和网络的输出 a,如图 7.36 所示,检测网络的性能. 为更方便比较给出上述两者的误差曲线如图 7.37 所示.

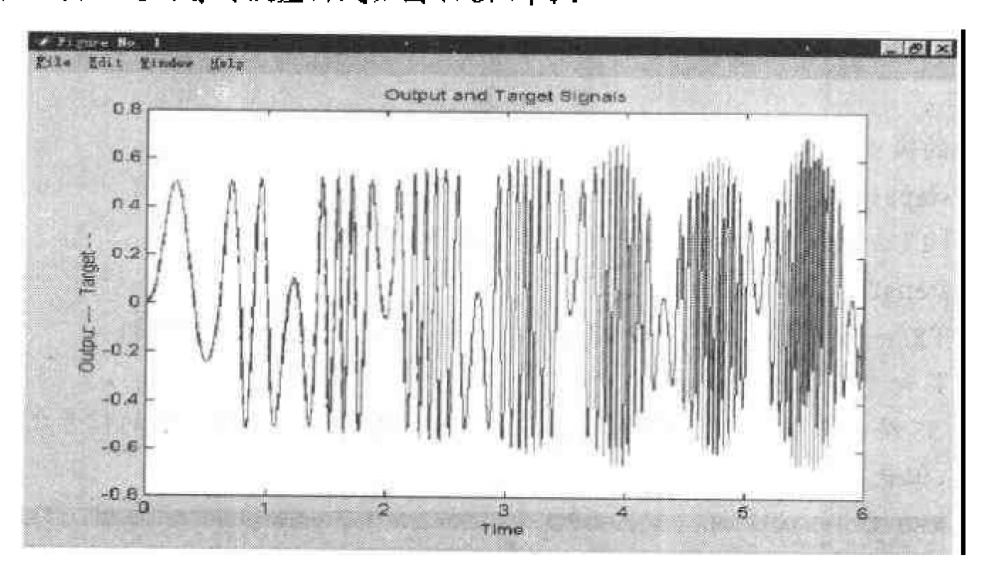


图 7.36 网络输出 a 和系统输出 t

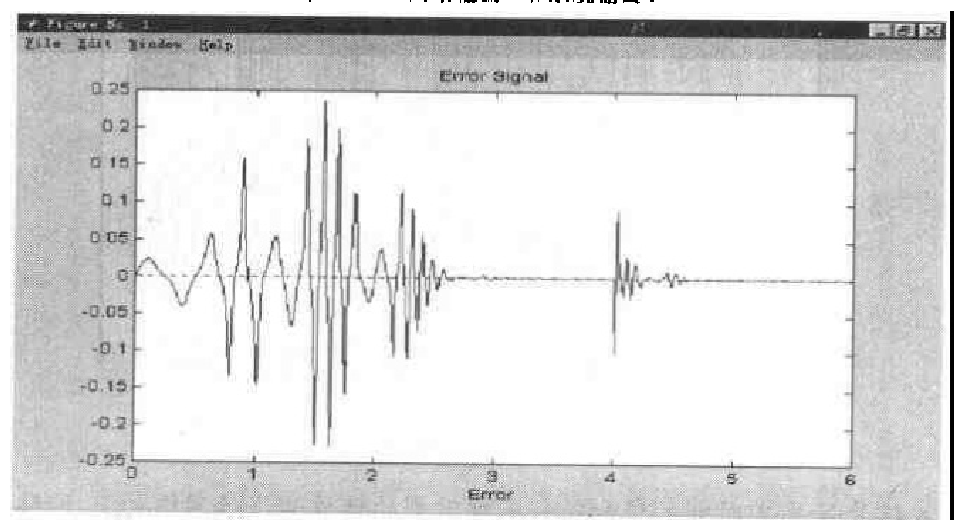


图 7.37 误差曲线

从误差曲线可见，在 2.5s 之后网络就能对系统进行非常准确的跟踪. 在第 4s 时，由于系统输入信号的突变，所以误差信号也发生了 0.5s 的波动，之后网络又可以精确地逼近线性系统.

5）自适应系统辨识的 MATLAB 程序

```
%APPLIN4
clf;
figure(gcf)
echo on
%      NEWLIN      建立一个线性网络
%      ADAPT     用 Widrow-Hoff 学习规则训练线性网络
pause % 按任意键继续
time1 = 0:0.005:4;            % from 0 to 4 seconds
time2 = 4.005:0.005:6;        % from 4 to 6 seconds
time = [time1 time2];         % from 0 to 6 seconds
X = sin(sin(time * 4) .* time * 8) ;
pause % 按任意键可求出系统的输出
steps1 = length(time1) ;
[T1,state] = filter([1 -0.5],1,X(1:steps1) );
steps2 = length(time2) ;
T2 = filter([0.9 -0.6],1,X((1:steps2) + steps1) ,state);
T = [T1 T2];
pause % 按任意键可以观察输入信号
plot(time,X)
xlabel('Time');
ylabel('Input Signal');
title('Input Signal to System');
pause % 按任意键可以观察系统的输出信号
plot(time,T)
xlabel('Time');
ylabel('Output Signal');
title('Output Signal of System');
pause % 按任意键定义网络的输入信号
T = con2seq(T);
P = con2seq(X);
pause % 按任意键开始设计网络
lr = 0.5;
delays = [0 1];
net = newlin(minmax(cat(2,P{:})),1,delays,lr);
pause % 按任意键开始对网络进行自适应训练
```

```
[net,y,e]=adapt(net,P,T);
pause % 按任意键可观察输出信号
plot(time,cat(2,y{:}),time,cat(2,T{:}),'--')
xlabel('Time');
ylabel('Output --- Target - -');
title('Output and Target Signals');
pause % 按任意键可观察误差信号
plot(time,cat(2,e{:}),[min(time) max(time)],[0 0],':r')
title('Time');
xlabel('Error');
title('Error Signal');
echo off
disp('End of APPLIN4')
```

第八章　BP　网　络

Rumelhart，McClelland 和他们的同事洞察到神经网络信息处理的重要性，于 1982 年成立了一个 PDP 小组，研究并行分布信息处理方法，探索人类认知的微结构. 1985 年发展了 BP 网络(Back-Propagation Network，简称 BP 网络)学习算法，实现了 Minsy 的多层网络设想.

目前，在人工神经网络的实际应用中，决大部分的神经网络模型是采用 BP 网络和它的变化形式，它也是前向网络的核心部分，并体现了人工神经网络最精华的部分.

BP 网络主要用于：

1）函数逼近：用输入矢量和相应的输出矢量训练一个网络逼近一个函数；

2）模式识别：用一个特定的输出矢量将它与输入矢量联系起来；

3）分类：把输入矢量以所定义的合适方式进行分类；

4）数据压缩：减少输出矢量维数以便于传输或存贮.

8.1　BP　网　络

8.1.1　BP 网络结构

1. 多层网络结构

多层 BP 网络不仅有输入节点、输出节点，而且有一层或多层隐节点，如图 8.1 所示.

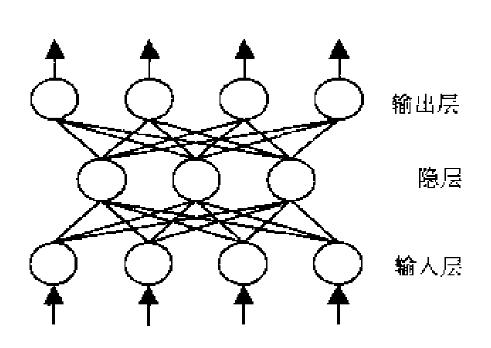


图 8.1　BP 网络模型结构

2. 传递函数一般为(0，1) S 型函数

$$f(x)=\frac{1}{1+e^{-x}}$$

3. 误差函数

对第 P 个样本误差计算公式为

$$E_P=\frac{\sum_i (t_{pi}-O_{pi})^2}{2}$$

式中 t_{pi},O_{pi}分别为期望输出和网络的计算输出.

8.1.2 BP 网络学习公式推导

网络学习公式推导的指导思想是,对网络权值(w_{ij},T_{li})的修正与阈值(θ)的修正,使误差函数(E)沿负梯度方向下降. BP 网络三层节点表示为,输入节点:x_j,隐节点:y_i,输出节点:O_l.

输入节点与隐节点间的网络权值为 w_{ij},隐节点与输出节点间的网络权值为 T_{li}. 当输出节点的期望输出为 t_l 时,BP 模型的计算公式如下:

(1) 隐节点的输出:

$$y_i=f(\sum_j w_{ij}x_j-\theta_i)=f(\mathrm{net}_i)$$

其中 $\mathrm{net}_i=\sum_j w_{ij}x_j-\theta_i$.

(2) 输出节点的计算输出:

$$O_l=f(\sum_i T_{li}y_i-\theta_l)=f(\mathrm{net}_l)$$

其中 $\mathrm{net}_l=\sum_i T_{li}y_i-\theta_l$.

(3) 输出节点的误差公式:

$$E=\frac{1}{2}\sum_l (t_l-O_l)^2=\frac{1}{2}\sum_l (t_l-f(\sum_i T_{li}y_i-\theta_l))^2$$

$$=\frac{1}{2}\sum_l (t_l-f(\sum_i T_{li}f(\sum_j w_{ij}x_j-\theta_i)-\theta_l))^2$$

1. 对输出节点的公式推导

$$\frac{\partial E}{\partial T_{li}}=\sum_{k=1}^{n}\frac{\partial E}{\partial O_k}\frac{\partial O_k}{\partial T_{li}}=\frac{\partial E}{\partial O_l}\frac{\partial O_l}{\partial T_{li}}$$

E 是多个 O_k 的函数,但只有一个 O_l 与 T_{li}有关,各 O_k 间相互独立. 其中

$$\frac{\partial E}{\partial O_l}=\frac{1}{2}\sum_k -2(t_k-O_k)\cdot\frac{\partial O_k}{\partial O_l}=-(t_l-O_l)$$

$$\frac{\partial O_l}{\partial T_{li}}=\frac{\partial O_l}{\partial \mathrm{net}_l}\frac{\partial \mathrm{net}_l}{\partial T_{li}}=f'(\mathrm{net}_l)\cdot y_i$$

则

$$\frac{\partial E}{\partial T_{li}}=-(t_l-O_l)\cdot f'(\mathrm{net}_l)\cdot y_i$$

设输入节点误差

$$\delta_l=-(t_l-O_l)\cdot f'(\mathrm{net}_l)$$

则

$$\frac{\partial E}{\partial T_{li}}=-\delta_l y_i$$

2. 对隐节点的公式推导

$$\frac{\partial E}{\partial w_{ij}}=\sum_{l}\sum_{i}\frac{\partial E}{\partial O_l}\frac{\partial O_l}{\partial y_i}\frac{\partial y_i}{\partial w_{ij}}$$

E 是多个 O_l 函数,针对某一个 w_{ij},对应一个 y_i,它与所有 O_l 有关(上式只存在对 l 的求和),其中

$$\frac{\partial E}{\partial O_l}=\frac{1}{2}\sum_{k}-2(t_k-O_k)\cdot\frac{\partial O_k}{O_l}=-(t_l-O_l)$$

$$\frac{\partial O_l}{\partial y_i}=\frac{\partial O_l}{\partial \text{net}_l}\cdot\frac{\partial \text{net}_l}{\partial y_i}=f'(\text{net}_l)\cdot\frac{\partial \text{net}_l}{\partial y_i}=f'(\text{net}_l)\cdot T_{li}$$

$$\frac{\partial y_i}{\partial w_{ij}}=\frac{\partial y_i}{\partial \text{net}_i}\cdot\frac{\partial \text{net}_i}{\partial w_{ij}}=f'(\text{net}_i)\cdot x_j$$

则

$$\frac{\partial E}{\partial w_{ij}}=-\sum_{l}(t_l-O_l)f'(\text{net}_l)\cdot T_{li}\cdot f'(\text{net}_i)x_j=-\sum_{l}\delta_l T_{li}\cdot f'(\text{net}_i)\cdot x_j$$

设隐节点误差

$$\delta_i'=f'(\text{net}_i)\cdot\sum_{l}\delta_l T_{li}$$

则

$$\frac{\partial E}{\partial w_{ij}}=-\delta_i' x_j$$

由于权值的修正 ΔT_{li},Δw_{ij}正比于误差函数沿梯度下降,则有

$$\Delta T_{li}=-\eta\frac{\partial E}{\partial T_{li}}=\eta\delta_l y_i$$

$$\delta_l=(t_l-O_l)\cdot f'(\text{net}_l)$$

$$\Delta w_{li}=-\eta'\frac{\partial E}{\partial w_{li}}=\eta'\delta_i x_j$$

$$\delta_i=f'(\text{net}_i)\sum_{l}\delta_l T_{li}$$

3. 基于公式汇总

1) 对输出节点:$\delta_l=(t_l-O_l)\cdot f'(\text{net}_l)$

2) 权值修正:$T_{li}(k+1)=T_{li}(k)+\Delta T_{li}=T_{li}(k)+\eta\delta_l y_i$

3) 对隐节点:$\delta=f'(\text{net}_i)\cdot\sum_{l}\delta_l T_{li}$

4) 权值修正:$w_{ij}(k+1)=w_{ij}(k)+\Delta w_{ij}=w_{ij}(k)+\eta'\delta_i' x_j$

其中隐节点误差 δ_i' 中的 $\sum_{l}\delta_l T_{li}$ 表示输出层节点 l 的误差 δ_l 通过权值 T_{li} 向隐节点 i 反向传播(误差 δ_l 乘权值 T_{li} 再累加)成为隐节点的误差,如图 8.2 所示.

4. 阈值的修正

阈值 θ 也是一个变化值,在修正权值的同时也修正它,原理同权值的修正一样.

(1) 对输出节点的公式推导

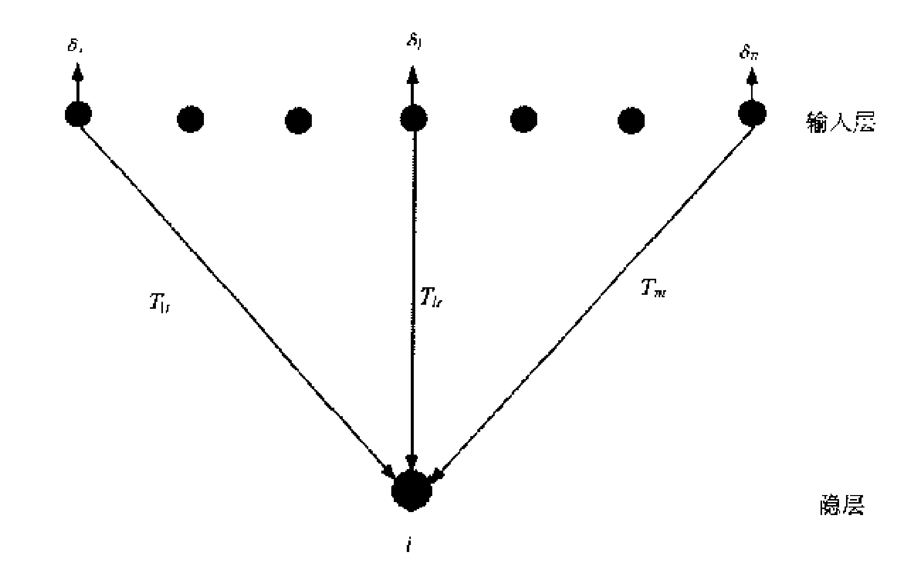


图 8.2 误差反向传播示意图

$$\frac{\partial E}{\partial \theta_l}=\frac{\partial E}{\partial O_l}\frac{\partial O_l}{\partial \theta_l}$$

其中

$$\frac{\partial E}{\partial O_l}=-(t_l-O_l)$$

$$\frac{\partial O_l}{\partial \theta_1}=\frac{\partial O_l}{\partial \mathrm{net}_l}\cdot\frac{\partial \mathrm{net}_l}{\partial \theta_l}=f'(\mathrm{net}_l)\cdot(-1)$$

则

$$\frac{\partial E}{\partial \theta_l}=(t_l-O_l)\cdot f'(\mathrm{net}_l)=\delta_l$$

由于

$$\Delta\theta_l=\eta\frac{\partial E}{\partial \theta_l}=\eta\delta_l$$

则

$$\theta_l(k+1)=\theta_l(k)+\eta\delta_l$$

(2) 对隐节点的公式推导

$$\frac{\partial E}{\partial \theta_i}=\sum_l\frac{\partial E}{\partial O_l}\frac{\partial O_l}{\partial y_i}\frac{\partial y_i}{\partial \theta_i}$$

其中

$$\frac{\partial E}{\partial O_l}=-(t_l-O_l)$$

$$\frac{\partial O_l}{\partial y_i}=f'(\mathrm{net}_l)\cdot T_{li}$$

$$\frac{\partial y_i}{\partial \theta_i}=\frac{\partial y_i}{\partial \mathrm{net}_i}\cdot\frac{\partial \mathrm{net}_i}{\partial \theta_i}=f'(\mathrm{net}_i)\cdot(-1)=\quad f'(\mathrm{net}_i),则$$

$$\frac{\partial E}{\partial \theta_i}=-\sum_l(t_l-O_l)f'(\mathrm{net}_l)\cdot T_{li}\cdot f'(\mathrm{net}_i)=\sum_l\delta_lT_{li}\cdot f'(\mathrm{net}_i)=\delta'_i$$

由于

$$\Delta\theta_l=\eta'\frac{\partial E}{\partial\theta_l}-\eta'\delta'_l$$

则

$$\theta_l(k+1)=\theta_l(k)+\eta'\delta'_l$$

5. 传递函数 $f(x)$ 的导数公式

函数 $f(x)=\frac{1}{1+e^{-x}}$，存在关系 $f'(x)=f(x)\cdot(1-f(x))$

则

$$f'(\text{net}_k)=f(\text{net}_k)\cdot(1-f(\text{net}_k))$$

对输出节点

$$O_l=f(\text{net}_l)$$
$$f'(\text{net}_l)=O_l\cdot(1-f(O_l))$$

对隐节点

$$y_i=f(\text{net}_i)$$
$$f'(\text{net}_i)=y_i\cdot(1-f(y_i))$$

6. BP 模型计算公式汇总

(1) 输出节点的输出 O_l 计算公式

1) 输入节点的输入：x_j

2) 隐节点的输出：$y_i=f(\sum_j w_{ij}X_j-\theta_i)$

其中连接权值 w_{ij}，节点阈值 θ_i.

3) 输出节点输出：$O_l=f(\sum_i T_{lj}y_i-\theta_l)$

其中连接权值 T_{lj}，节点阈值 θ_l.

(2) 输出层(隐节点到输出节点间)的修正公式

1) 输出节点的期望输出：t_l

2) 误差控制.

所有样本误差：

$E=\sum_{k=1}^{P}e_k<\varepsilon$，其中一个样本误差

$$e_k=\sum_{l=1}^{n}|t_l^{(k)}-O_l^{(k)}|$$

其中 P 为样本数，n 为输出节点数.

3) 误差公式：$\delta_l=(t_l-O_l)\cdot O_l\cdot(1-O_l)$

4) 权修正值：$T_{li}(k+1)=T_{li}(k)+\eta\delta_l y_i$

其中 k 为迭代次数.

5) 阈值修正：$\theta_l(k+1)=\theta_l(k)+\eta'\delta'_l$

(3) 隐节点层(输入节点到隐节点数)的修正公式

1）误差公式：$\delta_i' = y_i(1-y_i)\sum_l \delta_l T_{li}$

2）权值修正：$w_{ij}(k+1) = w_{ij}(k) + \eta' \delta_i' x_j$

3）阈值修正：$\theta_i(k+1) = \theta_i(k) + \eta' \delta_i'$

8.2 MATLAB 神经网络工具箱中的 BP 网络

8.2.1 BP 网络中的神经元模型

图 8.3 给出一个基本的神经元模型，它具有 R 个输入，每个输入都通过一个适当的权值 w 与神经元相连，神经元的输出可表示成

$$a = f(w * p, b)$$

BP 网络中基本神经元的激活函数必须处处可微，所以，经常使用的是 S 型的对数或正切激活函数或线性函数.

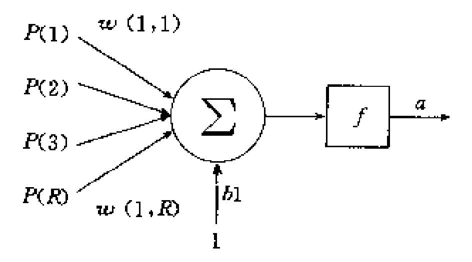


图8.3 基本的神经元模型

8.2.2 BP 网络结构

典型的 BP 网络结构如图 8.4 所示.

BP 网络通常有一个或多个隐层，隐层中的神经元均采用 S 型变换函数，输出层的神经元采用纯线性变换函数. 图 8.4 描述了一个具有一个隐层的 BP 网络.

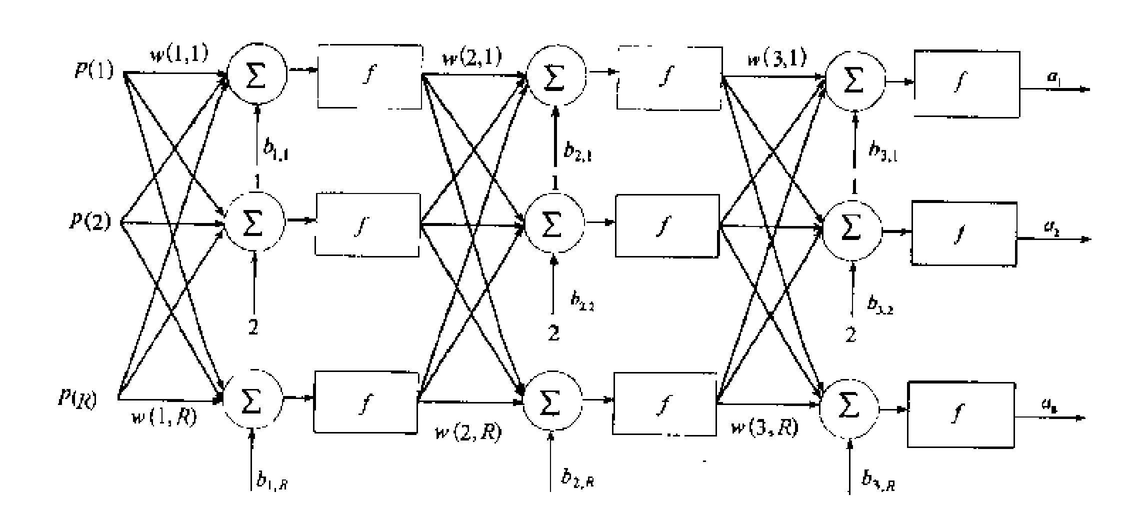


图 8.4 BP 网络结构

8.2.3 MATLAB 中有关 BP 网络的重要函数

MATLAB 的 BP 网络工具箱中包含了进行 BP 网络分析和设计的许多工具函数. 表 8.1 给出了这些函数的名称和基本功能(本章是在 MATLAB5.1 版本的基础上进行讨论, 有关高版本的这部分内容, 在理解本章内容的基础上, 结合 help 命令, 可顺利解决).

表 8.1 BP 网络的重要函数和功能

函数名	功能
deltalin	Purelin 神经元的 δ(delta)函数
deltalog	Logsig 神经元的 δ 函数
deltatan	Tansig 神经元的 δ 函数
errsurf	计算误差曲面
initff	至多三层的前向网络初始化
learnbp	反向传播学习规则
learnbpm	利用冲量规则的改进 BP 算法
learnlm	Levenberg-Marquardt 学习规则
logsig	对数 S 型传递函数
nwlog	对 Logsig 神经元产生 Nguyen-Midrow 随机数
nwtan	对 Togsig 神经元产生 Nguyen-Midrow 随机数
purelin	线性传递函数
simuff	前向网络仿真
tansig	正切 S 型传递函数
trainbp	利用 BP 算法训练前向网络
trainbpx	利用快速 BP 算法训练前向网络
trainlm	利用 Levenberg-Marquardt 规则训练前向网络

1. 神经元上的传递函数

(1) purelin 利用 Widrow-Hoff 或 BP 算法训练的神经元的传递函数经常采用线性函数.

语法格式: purelin(n)

purelin(z,b)

purelin(p)

说明: 线性传递函数如图 8.5 所示.

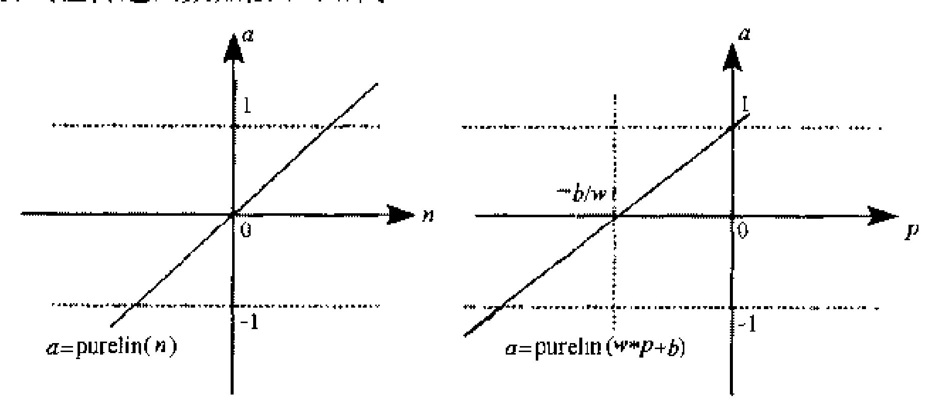


图 8.5 线性传递函数

神经元可采用的最简单的函数,即线性函数,它只是简单地将神经元输入经阈值调整后传递到输出.

purelin(n) 函数可得到输入矢量为 n 时网络层输出矩阵.

purelin(z,b) 函数可用于成批处理矢量,并且提供阈值的情况. 这时阈值矢量 b 与加权输入距阵 z 是区分的. 阈值矢量 b 加到 z 的每个矢量中去,以形成网络的输入矩阵,调用后得到的结果为矩阵.

purelin(p) 函数中,p 指出传递函数特性的名称,调用后可得到所询问的特性, 即:

purelin('delta') 指出 delta 函数名称.

purelin('init') 指出标准初始化函数名称.

purelin ('name')指出传递函数的全称.

purelin ('output')指出包含传递函数最大和最小输出值的二元矢量.

例 8.1 运行如下程序,可得如图 8.6 所示的结果.

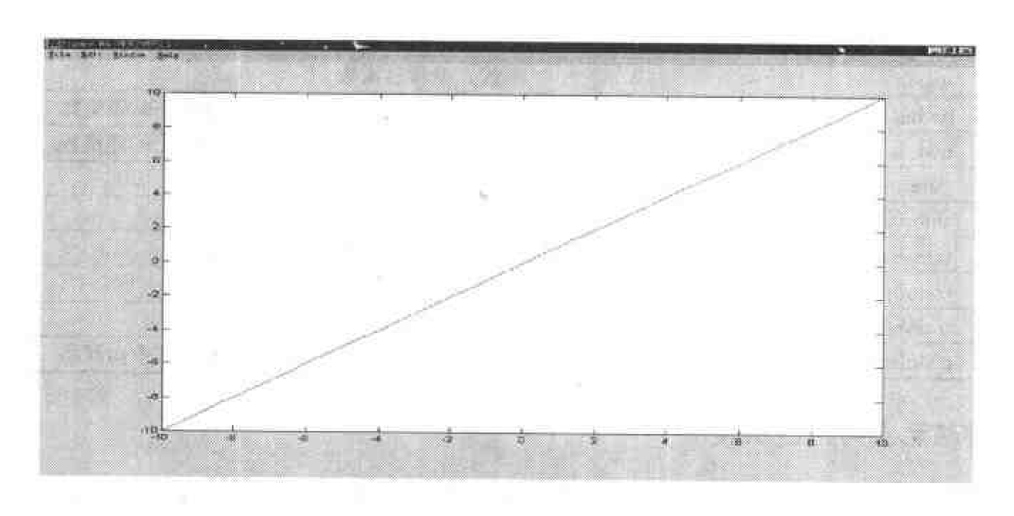


图 8.6 范例 8.1 的运行结果

```
n = -10:0.1:10;
a = purclin(n);
plot(n,a)
```

例 8.2 网络输入矢量为

```
n=[0.1 0.4 -1.2]
```

则纯线性传递函数神经元的输出为

```
a=purelin(n)
a =
      0.1000     0.4000     -1.2000
```

(1) tansig 双曲正切 S 型(sigmoid)传递函数

语法格式:tansig(n)

tansig(z,b)

tansig(p)

说明:双曲正切 S 型传递函数如图 8.7 所示.

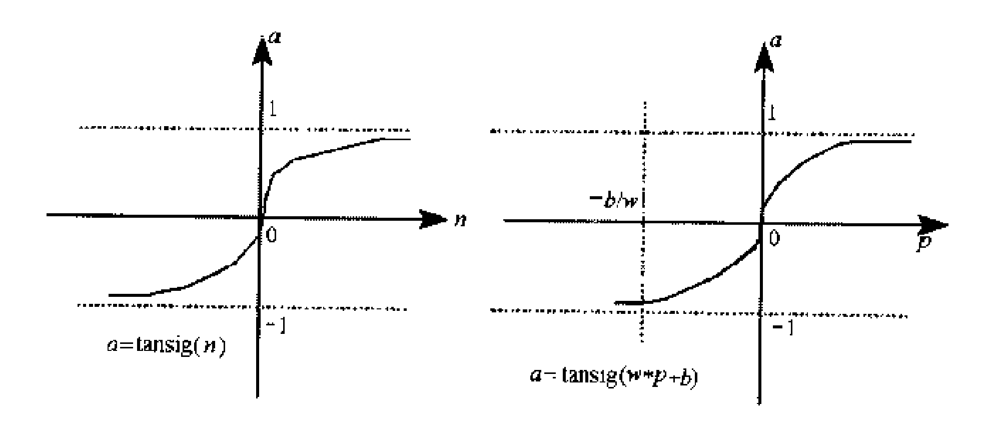


图 8.7　双曲正切S型传递函数

双曲正切 sigmoid 函数用于将神经元的输入范围为(−∞,+∞)映射到(−1,+1)．正切 sigmoid 函数是可微函数，因此很适合于利用 BP 算法训练神经网结. 函数表达式为

$$\mathrm{tansig}(n)=\tanh(n)$$

tansig(n) 函数对网络输入矢量 n 的所有元素求正切 sigmoid 函数.

tansig(z,b) 函数用于成批处理矢量，并且提供阈值的情况，这时阈值 b 与加权输入矩阵 z 是区分的，阈值矢量 b 加到 z 的每个矢量中去，以形成网络的输入矩阵，然后利用正切 sigmoid 函数将网络输入转换成输出.

tansig(p) 函数中，p 指出传递函数特性的名称，调用后可得到所询问的特性，即

tansig('delta') 指出 delta 函数名称.

tansig ('init') 指出标准初始化函数名称.

tansig ('name') 指出传递函数的全称.

tansig ('output') 指出包含传递函数最大和最小输出值的二元矢量.

例 8.3　运行如下程序，可得如图 8.8 所示的结果.

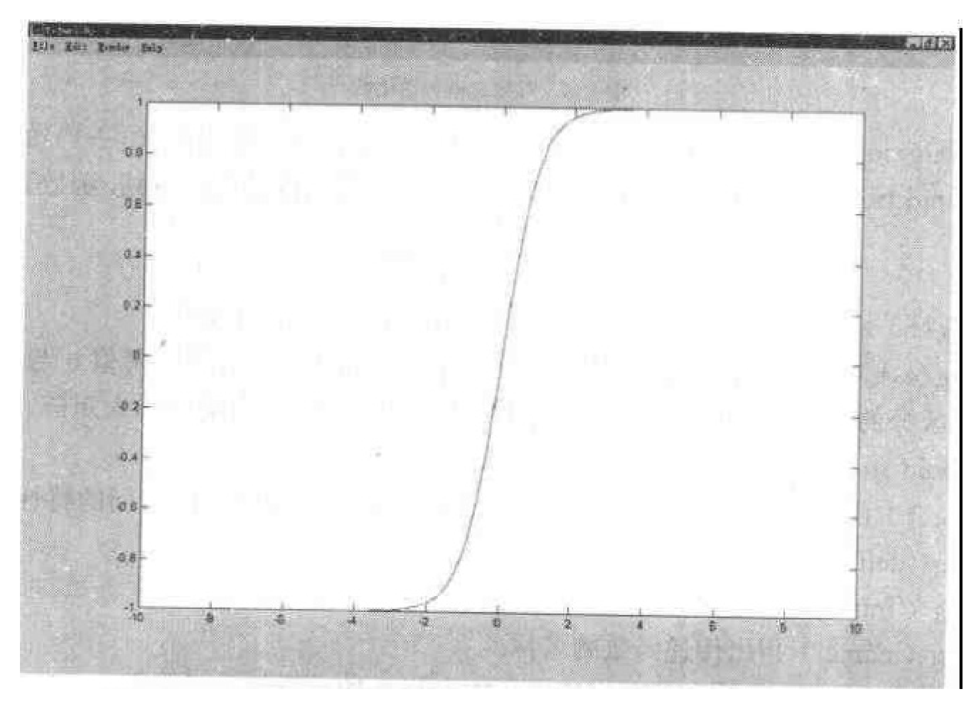


图 8.8　范例 8.3 的运行结果

```
n=-10:0.1:10;
a=tansig(n);
plot(n,a)
```

例 8.4　网络输入矢量为

```
n=[2.1  -1.0  0.2  1.0  2.0];
```

则双曲正切传递函数神经元的输出为

```
a=tansig(n)
a =
    0.9705   -0.7616   0.1974   0.7616   0.9640
```

(2) logsig　对数 S 型(sigmoid)传递函数.

语法格式:logsig(n)

　　logsig(z,b)

　　logsig(p)

说明:sigmoid 传递函数如图 8.9 所示.

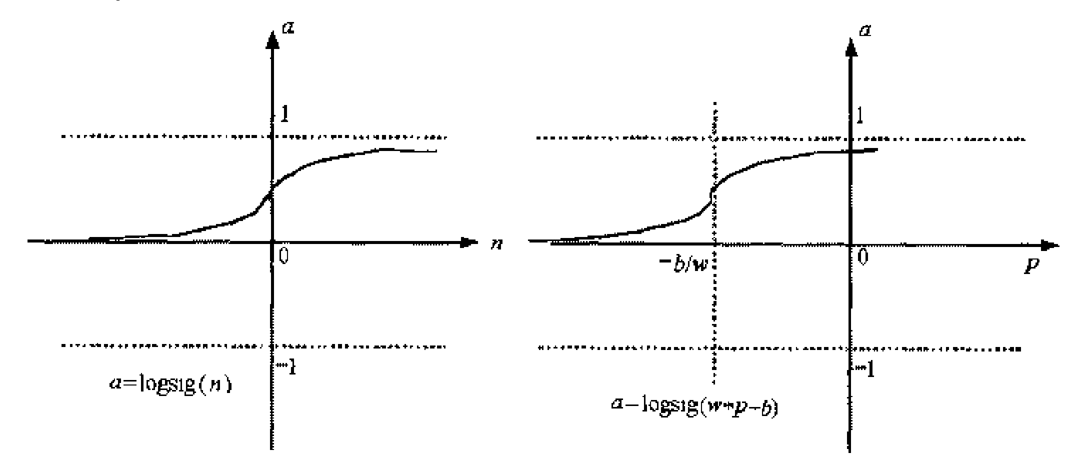


图 8.9　对数 S 型传递函数

对数 sigmoid 函数用于将神经元的输入范围(−∞,+∞)映射到(0,+1) 的区间上,对数 sigmoid 函数是可微函数,因此很适合于利用 BP 算法训练神经网络. 函数表达式为

$$\text{logsig}(n)=\frac{1}{1+e^{-n}}$$

logsig(n) 函数对网络输入阵 n 的所有元素求对数 logmoid 函数.

logsig(z,b) 函数用于成批处理矢量,并且提供阈值的情况,这时阈值 b 与加权输入矩阵 z 是区分的,阈值矢量 b 加到 z 的每个矢量中去,以形成网络的输入矩阵,然后利用对数 sigmoid 函数将网络输入转换成输出.

logsig(p) 函数中,p 指出传递函数特性的名称,调用后可得到所询问的特性,即

logsig('delta') 指出 delta 函数名称.

logsig ('init') 指出标准初始化函数名称.

logsig ('name') 指出传递函数的名称.

logsig ('output') 指出包含传递函数最大和最小输出值的二元矢量.

例 8.5 运行如下程序,可得如图 8.10 所示的结果.

```
n = -10:0.1:10;
a = logsig(n);
plot(n,a)
```

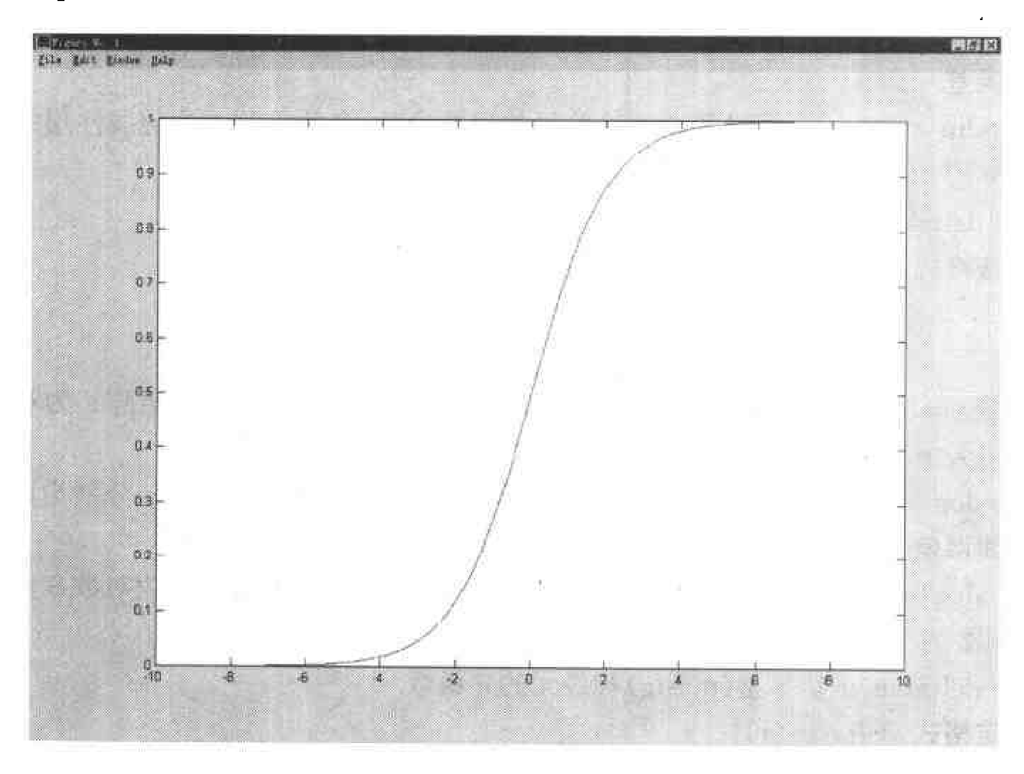


图 8.10 范例 8.5 的运行结果

例 8.6 网络输入阵为

n=[-1.0 2.1 0.3 0.0]';

则对数传递函数神经元的输出为

```
a=logsig(n)
a =
    0.2689
    0.8909
    0.5744
    0.5000
```

2. δ(Delta)函数

(1) deltalin 纯线性(purelin)神经元的 δ 函数.

语法格式:deltalin(a)

deltalin(a,e)

deltalin(a,d2,w2)

说明:通常的反向传播算法(BP)是利用网络误差平方和对网络各层输入的导数来调整其权值和阈值,从而降低误差平方和.从网络误差矢量中可推导出输出层的误差导数或

δ(delta)矢量，隐层的 δ 矢量可由下一层的 δ 矢量导出，这种 δ 矢量的反向传播正是 BP 算法的由来.

deltalin(a) 可计算出这一层输出对本层输出的导数，参数矩阵 a 为纯线性层的输出矢量.

deltalin(a,e) 可计算出线性输出层的误差导数，参数矩阵 a 和 e 分别为该层的输出矢量和误差.

deltalin(a,d2,w2) 可计算出线性隐层的误差导数，参数矩阵 a 为纯线性层的输出矢量，d2 为下一层的 δ 矢量，w2 为与下一层的连接权值.

(2) deltalog 对数 S 型(logsig)神经元的 δ 函数.

语法格式：deltalog(a)

deltalog(a,e)

deltalog(a,d2,w2)

说明：deltalog(a) 可计算出这一层输出对本层输出的导数，参数矩阵 a 为对数 S 型层的输出矢量.

deltalog(a,e) 可计算出 logsig 输出层的误差导数，参数矩阵 a 和 e 分别为该层的输出矢量和误差.

deltalog(a,d2,w2) 可计算出 logsig 隐层的误差导数，参数矩阵 a 为对数 S 型层的输出矢量，d2 为下一层的 δ 矢量，w2 为与下一层的连接权值.

(3) deltatan 正切 S 型(tansig)神经元的 δ 函数.

语法格式：deltatan(a)

deltaltan(a,e)

deltaltan(a,d2,w2)

说明：deltatan(a) 可计算出这一层输出对本层输出的导数，参数矩阵 a 为正切 S 型层的输出矢量.

deltatan(a,e) 可计算出 tansig 输出层的误差导数，参数矩阵 a 和 e 分别为该层的输出矢量和误差.

deltatan(a,d2,w2) 可计算出 tansig 隐层的误差导数，参数矩阵 a 为正切 S 型层的输出矢量.

3. 基本函数

(1) initff 前向网络初始化.

语法格式：[w,b]=initff(p,S,f)

[w1,b1,w2,b2]=initff(p,S1,f1,S2,f2)

[w1,b1,w2,b2,w3,b3]=initff(p,S1,f1,S2,f2,S3,f3)

[w,b]=initff(p,S,t)

[w1,b1,w2,b2]=initff(p,S1,f1,S2,t2)

[w1,b1,w2,b2,w3,b3]=initff(p,S1,f1,S2,f2,S3,t3)

说明：initff(p,S,f) 可得到 S 个神经元的单层神经网络的权值和阈值，其中 p 为输入矢量，f 为神经网络层间神经元的传递函数.

值得注意:p 中的每一行中必须包含网络期望输入的最大值和最小值,这样才能合理地初始化权值和阈值.

initff 可对至多三层神经网络进行初始化,可得到每层的权值及阈值.

此外,initff 也可用目标矢量 t 代替网络输出层的神经元数,这时输出层的神经元数目就为 t 的行数.

例 8.7　考虑一两层 BP 网络,网络的第一层由 4 个神经元组成,第二层由 2 个神经元组成,两输入矢量 p 定义了输入到网络的最大值和最小值:

p=[0 10; −5 5]

两层神经元的传递函数分别为"正切 S 型函数"和"线性函数",则其网络的初始权值和阈值可用 initff 函数产生,如图 8.11 所示.

图 8.11　范例 8.7

(2) simuff 前向网络仿真.

语法格式:simuff(p,w1,b1,f1)

simuff(p,w1,b1,f1,w2,b2,f2)

simuff(p,w1,b1,f1,w2,b2,f2,w3,b3,f3)

说明:前向网络由一系列网络层组成,每一层都从前一层得到输入数据,simuff 函数可仿真至多三层的前向网络.

例 8.8　先利用 initff 函数产生两层 Tansig/Purelin 前向网络的初始权值和阈值.

[w1,b1,w2,b2] = initff([0 10; −5 5],3,'tansig',2,'purelin');

当网络输入 p = [2; −3] 时;其响应可利用 simuff 计算得到,上述过程如图 8.12 所示,读者可在计算中进一步体会.

(3) trainbp 利用 BP 算法训练前向网络.

语法格式:[w,b,te,tr]=trainbp(w,b,'f',p,t,tp)

[w1,b1,w2,b2,te,tr]=trainbp(w1,b1,'f1',w2,b2,'f2',p,t,tp)

[w1,b1,w2,b2,,w3,b3,te,tr]=
　　trainbp(w1,b1,'f1',w2,b2,'f2',w3,b3,'f3',p,t,tp)

```
[w1,b1,w2,b2] = initff([0 10; -5 5],3,'tansig',2,'purelin')

w1 =

    0.2424   -0.0075
   -0.1373    0.1999
    0.0915    0.2245

b1 =

   -1.3174
   -0.4018
    0.3217

w2 =

   -0.1106    0.5839    0.4764
    0.2309    0.0436   -0.6475

b2 =

   -0.1886
    0.3709

p = [2; -3];
a = simuff(p,w1,b1,'tansig',w2,b2,'purelin')

a =

   -0.7053
    0.0860
```

图 8.12　范例 8.8

说明：利用 BP 算法训练前向网络，使网络完成函数逼近、矢量分类及模式识别.

[w,b,te,tr]=trainbp(w,b,'f',p,t,tp) 利用单层神经元连接的权值矩阵 w，阈值矢量 b 及传递函数名 f 成批训练网络，使当输入矢量为 p 时，网络的输出为目标矢量矩阵 t.

tp 为可选训练参数，其作用是设定如何进行训练，具体如下：

tp(1)　　显示间隔次数，其缺省值为 25；

tp(2)　　最大循环次数，其缺省值为 100；

tp(3)　　目标误差，其缺省值为 0.02；

tp(4)　　学习速率，其缺省值为 0.01.

值得注意，当指定了 tp 参数时，任何缺省或 NaN 值都会自动取其缺省值.

一旦训练达到了最大的训练次数，或者网络误差平方和降到期望误差之下时，都会使网络停止学习. 学习速率会影响权值与阈值更新的比例，较小的学习速率会导致学习时间增长，但有时可避免训练过程发散.

调用 trainbp 函数可得到新的权值矩阵 w、阈值矢量 b、网络训练的实际训练次数 te 及网络训练误差平方和的行矢量 tr.

对于多层网络，在调用 trainbp 函数时，可得到各层的权值矩阵及各层的阈值矢量.

(4) trainbpx 利用快速 BP 算法训练前向网络

语法格式：[w,b,te,tr]=trainbpx(w,b,'f',p,t,tp)

[w1,b1,w2,b2,te,tr]=trainbpx(w1,b1,'f1',w2,b2,'f2',p, t,tp)

[w1,b1,w2,b2,,w3,b3,te,tr]

=trainbpx(w1,b1,'f1',w2,b2,'f2',w3,b3,'f3',p,t,tp)

说明：当采用动量时,BP 算法可找到更优的解；当采用自适应学习速率时,BP 算法可缩短训练时间,trainbpx 函数利用这两种方法来训练多层前向网络.

[w,b,te,tr]=trainbpx(w,b,'f',p,t,tp) 利用单层神经元连接的权值矩阵 w,阈值矢量 b 及传递函数名 f 成批训练网络,使当输入矢量为 p 时,网络的输出为目标矢量矩阵 t.

tp 为可选训练参数，其作用是设定如何进行训练,具体如下：

tp(1)　设定显示间隔次数,其缺省值为 25；

tp(2)　设定最大循环次数，其缺省值为 100；

tp(3)　设定目标误差，其缺省值为 0.02；

tp(4)　设定学习速率，其缺省值为 0.01；

tp(5)　设定学习速率增加的比率，其缺省值为 1.05；

tp(6)　设定学习速率减少的比率，其缺省值为 0.7；

tp(7)　设定动量常数，其缺省值为 0.9；

tp(8)　设定最大误差比率，其缺省值为 1.04.

值得注意,当指定了 tp 参数时,任何缺省或 NaN 值都会自动取其缺省值.

一旦训练达到了最大的训练次数,或者网络误差平方和降到期望误差之下时,都会使网络停止学习.自适应学习速率先给一个初值,然后利用乘法使之增加或减小,以保持学习速度快而且稳定.定义在 0 到 1 之间的动量指定了所使用的动量大小,误差比率则限制了单次训练中可能增加的误差,如果误差上升超过了误差比率,则舍弃新的权值,并暂时不使用动量.

调用 trainbpx 函数可得到新的权值矩阵 w、阈值矢量 b、网络训练的实际训练次数 te 及矩阵 tr,tr 的第一行为训练过程中网络的误差,第二行为相应的自适应学习速率.当以二层或三层网络的权值矩阵、阈值矢量及传递函数调用 trainbpx 函数时,可得到各层的权值矩阵及各层的阈值矢量.

(5) trainelm 训练 Elman 递归网络

语法格式：[w1,b1,w2,b2,te,tr]=trainbpx(w1,b1,w2,b2,p,t,tp)

说明：Elman 网络由一个正切 S 型隐层和一个纯线性输出层组成.tansig 层接收网络输入及从自身的反馈信号,纯线性层从 Tansig 层得到输入.由于 Elman 网络为 S 型/线性网络,因此它能实现任何有限值函数.由于它有反馈连接,因此通过学习可识别或产生时间模式和空间模式.

[w1,b1,w2,b2,te,tr]=trainbpx(w1,b1,w2,b2,p,t,tp) 函数中,w1,b1 为 tansig 层的权值和阈值,w2,b2 为线性输出层的权值和阈值,p 为输入矢量,t 为相应的目标矢量,网络采用快速 BP 算法训练,以便产生相应于矢量 p 的输出矢量 t .

tp 为可选训练参数，其作用是设定如何进行训练,具体如下：

tp(1)　设定显示间隔次数,其缺省值为 5；

tp(2)　设定最大循环次数，其缺省值为 500；

tp(3)　设定目标误差，其缺省值为 0.01；

tp(4)　设定初始自适应学习速率，其缺省值为 0.001；

tp(5)　设定学习速率增加的比率，其缺省值为 1.05；

tp(6)　设定学习速率减少的比率，其缺省值为 0.7；

tp(7)　设定动量常数，其缺省值为 0.95；

tp(8)　设定误差比率，其缺省值为 1.04.

值得注意，当指定了 tp 参数时，任何缺省或 NaN 值都会自动取其缺省值.

(6) trainlm 利用 Levenberg-Marquardt 规则训练前向网络

语法格式：[w,b,te,tr]=trainlm(w,b,'f',p,t,tp)

[w1,b1,w2,b2,te,tr]=trainlm(w1,b1,'f1',w2,b2,'f2',p,t,tp)

[w1,b1,w2,b2,,w3,b3,te,tr]=

trainlm(w1,b1,'f1',w2,b2,'f2',w3,b3,'f3',p,t,tp)

说明：Levenberg-Marquardt 算法比 trainbp 和 trainbpx 函数使用的梯度下降法要快得多，但它需要更多的内存.

[w,b,te]=trainlm(w,b,,p,t,tp) 利用训练参数 tp，输入矢量矩阵 p 和相应目标矢量 t，对初始权值及阈值进行训练，从而得到新的权值 w 和阈值矢量 b.

tp 为可选训练参数，其作用是设定如何进行训练，具体如下：

tp(1)　设定显示间隔次数，其缺省值为 25；

tp(2)　设定最大循环次数，其缺省值为 1000；

tp(3)　设定目标误差，其缺省值为 0.02；

tp(4)　设定最小梯度，其缺省值为 0.0001；

tp(5)　设定 μ 的初始值，其缺省值为 0.001；

tp(6)　设定 μ 的增加系数，其缺省值为 10；

tp(7)　设定 μ 的减小系数，其缺省值为 0.1；

tp(8)　设定 μ 的最大值，其缺省值为 1e10.

值得注意，当指定了 tp 参数时，任何缺省或 NaN 值都会自动取其缺省值.

只有当训练中的误差达到了期望误差，μ 达到了最大值，或者达到了最大的训练次数，才会停止训练.

变量 μ 确定了学习算法是根据牛顿法还是梯度法来完成，下式为更新参数的 Levenberg-Marquardt 规则

$$\Delta w=(J^TJ+\mu I)^{-1}\cdot J^Te$$

随着 μ 的增大，Levenberg-Marquardt 中的项 J^TJ 可以忽略. 因此学习过程主要根据梯度下降，即 $\mu^{-1}J^Te$ 项. 只要迭代使误差增加，μ 也就会增加，直到误差不再增加为止，但是，如果 μ 太大，则会使学习停止(因 $\mu^{-1}J^Te$ 接近于 0)，当已经找到最小误差时，就会出现这种情况，这就是为什么当 μ 达到最小值时要停止学习的原因.

4. 学习规则函数

(1) learnbp 反向传播学习规则函数

语法格式：learnbp(p,d,lr)

[dw,db] = learnbp(p,d,lr)

说明：反向传播(BP)学习规则为调整网络的权值和阈值使网络误差的平方和最小，

这是通过在最速下降方向上不断地调整网络的权值和阈值来达到的.

首先计算出网络输出层的误差矢量的导数,然后通过网络反向传播,直到计算出每个隐层的误差导数(称为 δ 矢量),这可利用函数 deltalin,deltalog 和 deltatan 来计算.

根据 BP 准则,每一层的权值矩阵 w 利用本层的 δ 矢量 d 和输入矢量 p 来更新

$$\Delta w(i,j)=lr\cdot d(i)\cdot p(j)$$

其中 lr 为学习速率.

调用 learnbp(p,d,lr) 后,可得到权值修正矩阵,其中 p 为本层的输入矢量,d 为 δ 矢量,lr 为学习速率.

[dw,db] = learnbp(p,d,lr) 可同时得到权值修正矩阵和阈值修正矢量.

(2) learnbpm 利用动量规则的改进 BP 算法

语法格式:learnbpm(p,d,lr,mc,dw,db)

[dw,db] = learnbpm(p,d,lr,mc,dw,db)

说明:使权值的变化等于上次权值的变化与这次由 BP 准则引起的变化之和,这样可将动量加到 BP 学习中,上一次权值变化的影响可由动量常数来调节. 当动量常数为 0 时,说明权值的变化仅由梯度决定. 当动量常数为 1 时,说明新的权值的变化仅等于上次权值变化,而忽略掉梯度项,其数学表达式为

$$\Delta w(i,j)=mc\cdot \Delta w(i,j)+(1-mc)\cdot lr\cdot d(i)\cdot p(j)$$

其中 mc 为动量常数.

调用 learnbpm(p,d,lr,mc,dw,db) 后,可得到新权值修正矩阵,其中 p 为网络层的输入矢量,d 为 δ 矢量,lr 为学习速率,mc 为动量常数,dw 为上一次权值变化矩阵.

[dw,db] = learnbp(p,d,lr,mc,dw,db) 可同时得到新权值修正矩阵和新阈值修正矢量.

(3) learnlm Levenberg-Marquardt 学习规则

语法格式:learnlm(p,d)

说明:这一函数用于计算网络输出误差对网络层权值的导数,利用 Levenberg-Marquardt 算法训练前向网络时(trainlm),要用到这种计算. Levenberg-Marquardt 算法的训练速度比梯度下降法要快得多,但需要更多的内存.

在 learnlm(p,d)函数中,p 为网络层的输入矢量,d 为每个网络误差对网络层每个输入的导数的雅可比(Jacobian)矩阵,调用后可得到每个网络误差对网络层每个权值的导数的 Jacobian 矩阵.

5. 绘图函数

(1) plotes 绘制误差曲面图

语法格式:plotes(wv,bv,es)

plotes(wv,bv,es,v)

说明:plotes(wv,bv,es) 绘制出由权值 wv 和阈值 bv 确定的误差曲面图 es(由 errsurf 产生),误差曲面图以三维曲面和等高线图形式显示.

plotes(wv,bv,es,v) 允许设置期望的视角 v,缺省值为[-37.5 30].

(2) plotep 在误差曲面上绘制出权值和阈值的位置

语法格式:plotep(w,b,e)

　　　　plotep(w,b,e,h)

说明: plotep(w,b,e) 在由 plotes 绘制的误差曲面图上,绘制出单输入神经元误差为 e 时的权值 w 和阈值 b 的位置.

当带输出变量调用时,plotep 函数可返回包含有关位置信息的矢量 h. plotep(w,b,e,h)可利用 h 清除网络前一次位置,并重新画出新的神经元位置.

6. 误差分析函数

errsurf 计算误差曲面

语法格式:errsurf(p,t,wv,bv,f)

说明: errsurf(p,t,wv,bv,f) 可计算单输入神经元误差曲面的平方和. 输入为输入行矢量 p,相应的目标 t 和传递函数名 f. 误差曲面是从每个权值与由权值 wv,阈值 bv 的行矢量确定的阈值的组合中计算出来的.

例 8.9　输入矢量 p 和相应的目标矢量 t 分别为

```
p = [-6.0 -6.1 -4.1 -4.0 +4.0 +4.1 +6.0 +6.1];
t = [+0.0 +0.0 +.97 +.99 +.01 +.03 +1.0 +1.0];
```

权值取为(−1,+1) 之间,阈值取为(−2.5,+2.5) 之间,以此来计算对数 S 型神经元的误差曲面的程序如下:

```
wv = -1:.1:1; bv = -2.5:.25:2.5;
ES = errsurf(p,t,wv,bv,'logsig');
plotes(wv,bv,ES,[60 30])
```

其程序的运行结果如图 8.13 所示.

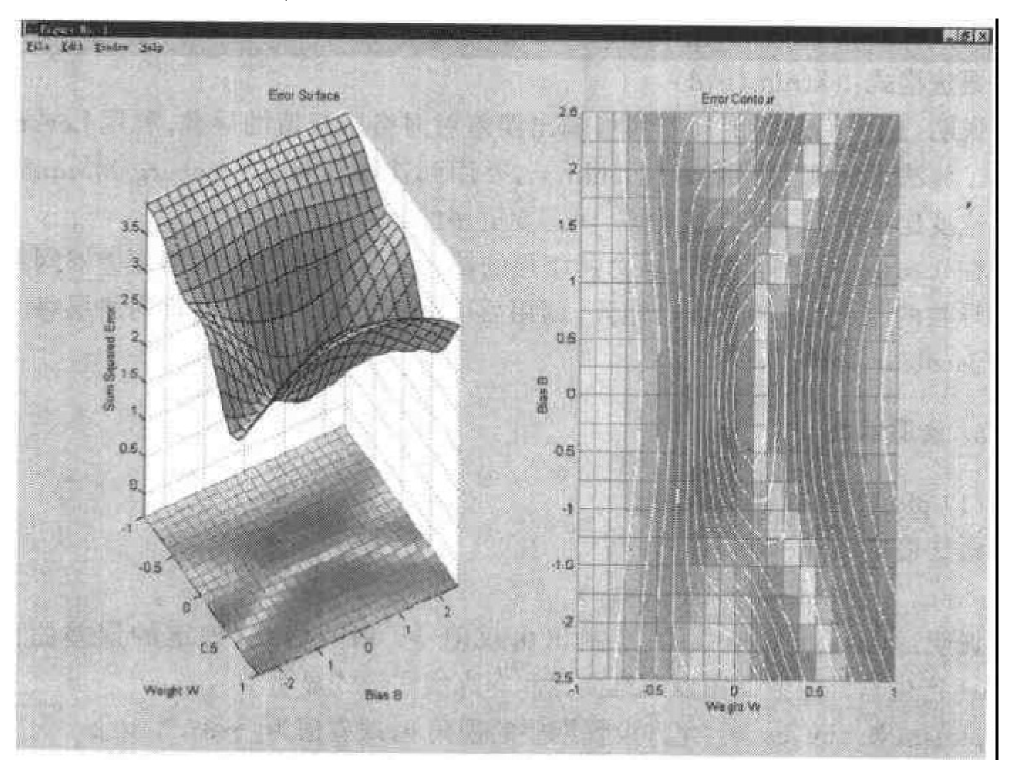


图 8.13　范例 8.9

8.3　BP 网络的应用设计

8.3.1　BP 网络的训练过程

为了应用神经网络，在选定所要设计的神经网络的结构之后（其中包括的内容有：网络层数、每层所含神经元的个数和神经元的激活函数），首先应考虑神经网络的训练过程．下面用两层网络为例来叙述 BP 网络的训练步骤．

步骤 1：用小的随机数对每一层的权值 w 和偏差 b 初始化，以保证网络不被大的加权输入饱和，同时还要进行以下参数的设定或初始化：

1）设定期望误差最小值：err _ goal；

2）设定最大循环次数：max _ epoch；

3）设置修正权值的学习速率：一般选取 lr＝0.01～0.7；

4）从 1 开始的循环训练：for epoch＝1：max _ epoch.

步骤 2：计算网络各层输出矢量 A1 和 A2 以及网络误差 E：

```
A1=tansig(w1*p,b1) ;
A2=purelin(w2*A1,b2) ;
E=T-A;
```

步骤 3：计算各层反向传播的误差变化 D2 和 D1，并计算各层权值的修正值以及新的权值：

```
D2=deltalin(A2,E);
D1=deltalan(A1,D2,w2) ;
[dw1,db1]=learnbp(p,D1,lr);
[dw2,db2]=learnbp(A1,D2,lr);
w1=w1+dw1;b1=b1+db1;
w2=w2+dw2;b2=b2+db2;
```

步骤 4：再次计算权值修正后的误差平方和：

```
SSE=sumsqr(T-purelin(w2*tansig(w1*p,b1) ,b2) );
```

步骤 5：检查 SSE 是否小于 err _ goal，若是，训练结束否则继续．

以上就是 BP 网络利用 MATLAB 神经网络工具箱训练的过程．以上所有的学习规则与训练的全过程，还可以用函数 trainbp 来代替．它的使用同样需要定义有关参数：显示间隔次数、最大循环次数、目标误差和学习速率，在调用 trainbp 函数后，返回训练后的权值、循环训练的总数和最终误差（关于 trainbp 的使用方法上节已介绍过）．

例 8.10　训练一非线性神经元，其结构如图 8.14 所示，并定义输入向量和目标向量分别为

```
p=[-3.0 2.0];
t=[0.4 0.8];
```

首先，利用函数 initff 对网络进行初始化

```
[w,b]=initff(p,t,'logsig');
```

然后，利用函数 trainbp 对网络进行训练

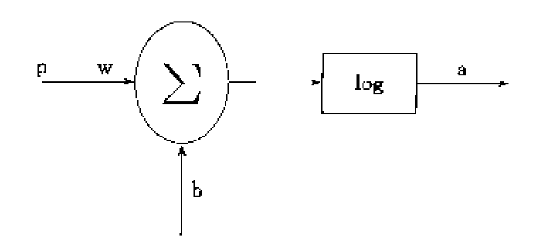


图8.14 非线性神经元结构

```
disp_fqre=1;max_epoch=100;err_goal=0.001;lr=2;
TP=[disp_fqre max_epoch err_goal lr];
[w,b,epochs,tr]=trainbp(w,b,'logsig',p,t,TP)
```

图 8.15 给出了 17 次循环后的最终训练结果，此时 SSE = 0.000895135，w = 0.3887.

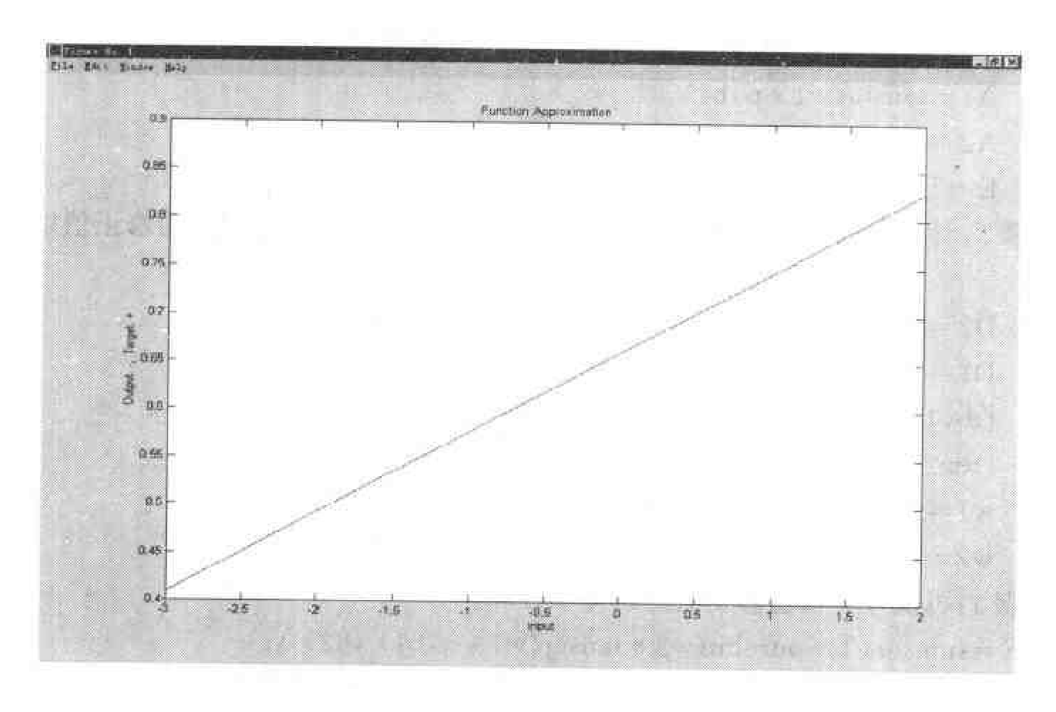


图 8.15 范例 8.10 最终训练结果

针对例 8.10，MATLAB 在神经网络工具箱中还给出了如下示范程序：

```
clf;
figure(gcf)
colordef(gcf,'none')
setfsize(500,200) ;
echo on
clc
% INITFF    对前向网络初始化
% TRAINBP    用 BP 算法训练前向网络
% SIMUFF    对前向网络仿真
```

```
pause % 键入任意键继续
clc
P = [-3.0 -2.0];
T = [+0.4 +0.8];
pause % 键入任意键继续可看到误差曲面
clc
wv = -4:0.4:4;
bv = -4:0.4:4;
es = errsurf(P,T,wv,bv,'logsig');
plotes(wv,bv,es,[60 30]);
pause % 键入任意键可设计网络
clc
w,b] = initff(P,T,'logsig')
echo off
k = pickic(1) ;
if k == 2
  w = -2.1617;
  b = -1.7862;
elseif k == 3
  subplot(1,2,2) ;
  h = centext(' * CLICK ON ME *');
  set(h,'color',[0 0 0]);
  [w,b] = ginput(1) ;
  delete(h)
end
echo on
clc
df = 5;      % 学习过程显示频率
me = 100;    % 最大训练循环次数
eg = 0.01;   %误差指标
lr = 2;      % 学习率
[w,b,ep,tr] = tbp1(w,b,'logsig',P,T,[df me eg lr],wv,bv,es,[60 30]);
pause        %键入任意键可看训练误差曲线
clc
ploterr(tr,eg);
pause
clc
p = -1.2;
a = simuff(p,w,b,'logsig')
```

```
echo off
disp('End of DEMOBP1')
```

例 8.11 用于函数逼近的 BP 网络设计.

设计一两层 BP 网络，其网络的隐层各神经元的激活函数为双曲正切型输出层各神经元的激活函数为线性函数，隐层含有 5 个神经元，并且有如下 21 组单输入矢量和相对应的目标矢量：

```
p=-1:0.1:1;
t=[-0.96 -0.577 -0.0729 0.377 0.641 0.66 0.461 0.1336 -0.201 -0.434
-0.5 - 0.393 - 0.1647 0.0988 0.3072 0.396 0.3449 0.1816 - 0.0312 - 0.2183
-0.3201];
```

图 8.16 给出了目标矢量 t 相对于输入矢量的图形. 下面考虑对网络进行训练.

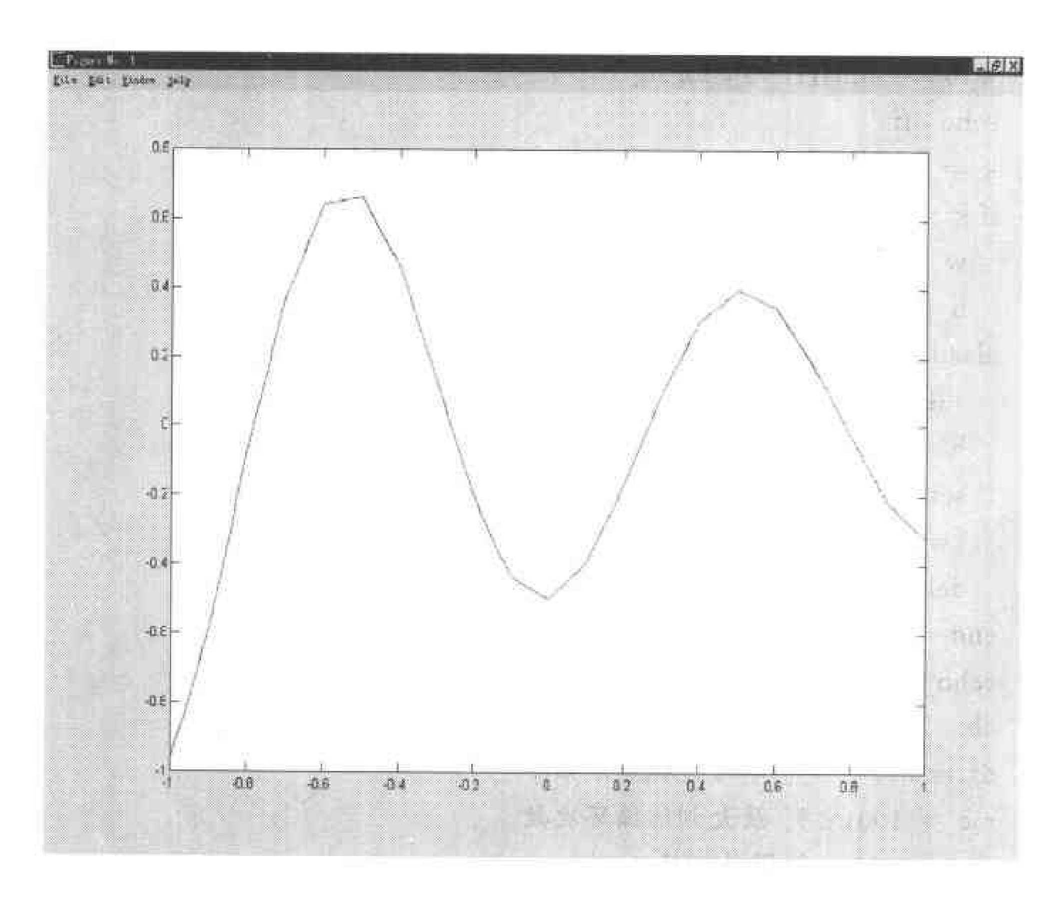


图 8.16 以目标矢量相对于输入矢量的图形

首先对网络进行初始化：

```
[R,Q]=size(p);
[S2,Q]=size(t);
S1=5;
[w1,b1]=rands(S1,R);
[w2,b2]=rands(S2,S1) ;
```

用输入矢量 p 计算网络的输出：

```
A2=purelin(w2 * tansig(w1 * p,b1) ,b2)
```

可以画出输出结果,观察初始网络是如何接近期望训练曲线的,如图 8.17 所示.

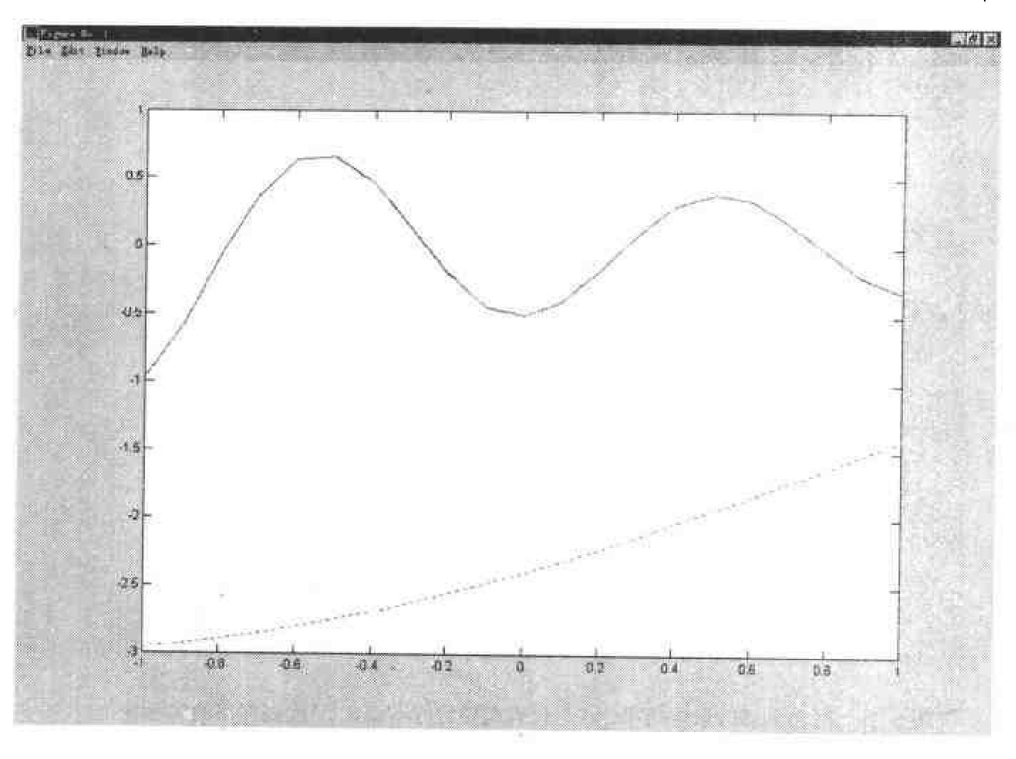


图 8.17 初始网络的输出曲线(初始网络的输出曲线用虚线表示)

下面定义网络训练参数:

```
disp _fqre=10;max _epoch=18000;err _goal=0.01;lr=0.01;
    TP=[disp _fqre max _epoch err _goal lr];
[w1,b1,w2,b2,epochs,errors]=trainbp(w1,b1,'tansig',w2,b2,'purelin',p,
t,TP)
```

运行上述程序后,可以返回训练后的权值、训练次数和误差平方和:

```
TRAINBP: 14161/18000 epochs, SSE = 0.00999891.
ans =
    -1.4293
    3.2465
    -0.3808
    2.7740
    2.3477
```

图 8.18 给出了 14161 次循环训练后的最终网络结果,网络的误差平方和落在所设定的 14161 以内(SSE = 0.00999891),图 8.19 给出了训练过程中的误差记录(此误差曲线可以用 ploterr(errors) 命令画出).

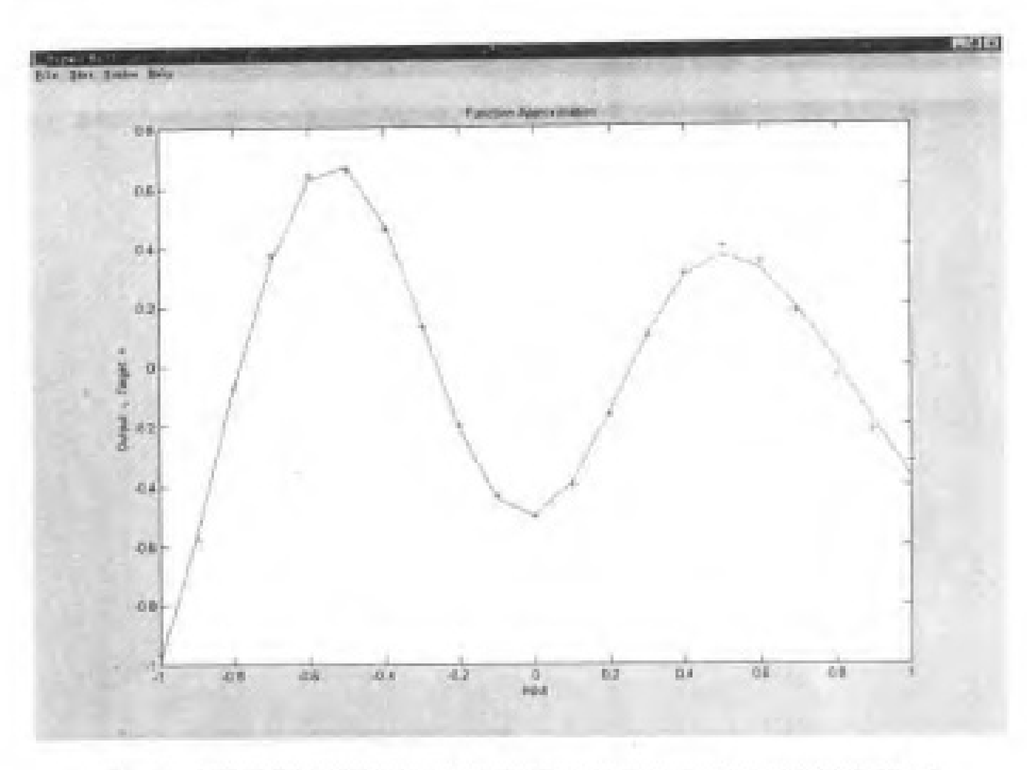


图 8.18　网络训练的最终结果('+' 代表期望的样本点，'.'代表网络的实际输出)

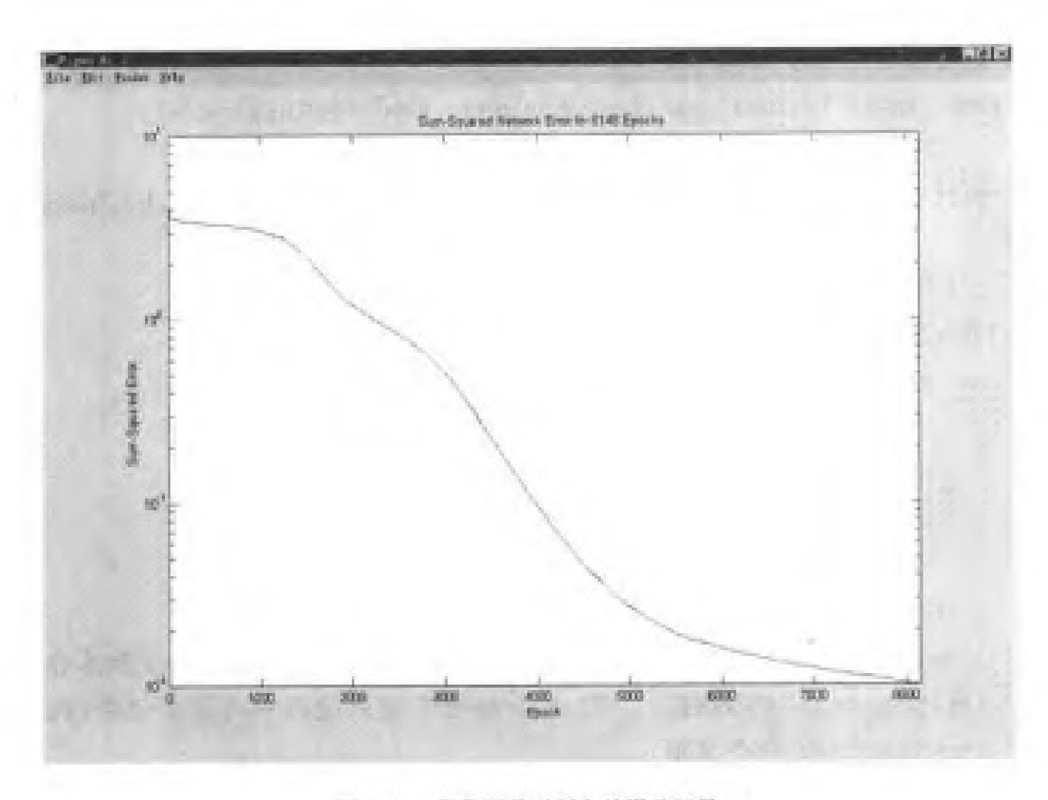


图 8.19　网络训练过程中的误差记录

8.3.2　BP 网络的设计分析

在进行 BP 网络设计前，一般应从网络的层数、每层中的神经元个数、初始值以及学习方法等方面来进行考虑. 下面首先分析 BP 网络的结构特征，然后结合 MATLAB 讨论一下相关的分析设计问题.

1. *网络的层数*

理论上早已证明：具有偏差和至少一个 S 型隐含层加上一个线性输出层的网络，能够逼近任何有理函数. 增加层数主要可以更进一步的降低误差，提高精度，但同时也使网络复杂化，从而增加了网络权值的训练时间. 而误差精度的提高实际上也可以通过增加隐含层中的神经元数目来获得，其训练效果也比增加层数更容易观察和调整，所以一般情况下，应优先考虑增加隐含层中的神经元数.

2. *隐含层的神经元数网络的层数*

网络训练精度的提高，可以通过采用一个隐含层，而增加其神经元个数的方法来获得. 这在结构实现上，要比增加更多的隐含层要简单得多. 那么究竟选取多少个隐含节点才合适？这在理论上并没有一个明确的规定. 在具体设计时，比较实际的做法是通过对不同神经元数进行训练比较对比，然后适当地加上一点余量.

3. *初始权值的选取*

由于系统是非线性的，初始值对于学习是否达到局部最小、是否能够收敛以及训练时间的长短关系很大. 如果初始值太大，使得加权后的输入落在激活函数的饱和区，从而导致其导数 $f'(x)$ 非常小，而在计算权值修正公式中，因为 δ 正比于 $f'(x)$，当 $f'(x)\to 0$ 时，则有 $\delta\to 0$. 使得 $w\to 0$，从而使得调节过程几乎停顿下来. 所以，一般总是希望经过初始加权后的每个神经元的输出值都接近于零，这样可以保证每个神经元的权值都能够在它们的 S 型激活函数变化最大之处进行调节. 所以，一般取初始权值在 $(-1,1)$ 之间的随机数. 另外，为了防止上述现象的发生，已有学者在分析了两层网络是如何对一个函数进行训练后，提出一种选定初始权值的策略：选择权值的量级为 $\sqrt[r]{S1}$，其中 $S1$ 为第一层神经元数目. 利用这种方法可以在较少的训练次数下得到满意的训练结果.

4. *学习速率*

学习速率决定每一次循环训练中所产生的权值变化量. 大的学习速率可能导致系统的不稳定，但小的学习速率将会导致训练较长，收敛速度很慢，不过能保证网络的误差值不跳出误差表面的低谷而最终趋于最小误差值. 所以在一般情况下，倾向于选取较小的学习速率以保证系统的稳定性. 学习速率的选取范围在 0.01～0.8 之间.

和初始权值的选取过程一样，在一个神经网络的设计中，网络要经过几个不同的学习速率的训练，通过观察每一次训练后的误差平方和 $\sum e^2$ 的下降速率来判断所选定的学习速率是否合适. 如果 $\sum e^2$ 下降很快，则说明学习速率合适，若 $\sum e^2$ 出现振荡现象，则说明

学习速率过大.对于一个具体网络都存在一个合适的学习速率.但对于较复杂网络,在误差曲面的不同部位可能需要不同的学习速率.为了减少寻找学习速率的训练次数以及训练时间,比较合适的方法是采用变化的自适应学习速率,使网络的训练在不同的阶段自动设置不同学习速率的大小.

5. 期望误差的选取

在设计网络的训练过程中,期望误差值也应当通过对比训练后确定一个合适的值,这个所谓的"合适",是相对于所需要的隐含层的节点数来确定,因为较小的期望误差值是要靠增加隐含层的节点,以及训练时间来获得的.一般情况下,作为对比,可以同时对两个不同期望误差值的网络进行训练,最后通过综合因素的考虑来确定采用其中一个网络.

8.4 BP 算法的改进及其设计实例

在实际应用中,原始的 BP 算法很难胜任,因此出现了很多的改进算法. BP 算法的改进主要有两种途径,一种是采用启发式学习方法,另一种则是采用更有效的优化算法.

在神经网络工具箱中,函数 trainbpx()采用动量法和学习率自适应调整两种策略,从而提高了学习速度并增加了算法的可靠性.动量法降低了网络对于误差曲面局部细节的敏感性,有效地抑制网络陷于局部极小;自适应调整学习率有利于缩短学习时间.

例 8.12 应用动量 BP 算法训练神经网络.

解 在应用动量对网络进行训练之前,首先要将网络权值和阈值的修正值矩阵初始化为零矩阵,即

```
dw=zeros(size(w));
db=zeros(size(b));
```

然后利用函数 learnbpm 求出修正矩阵

```
[dw,db]=learnbpm(p,d,lr,mc,dw,db)
```

其中 lr 是学习率,mc 是动量常数.由此得到网络权值和阈值的更新值

```
w=w+dw;
b=b+db;
```

利用函数 trainbpm 可完成对网络进行训练

```
[w,b,epochs,tr]=trainbpm(w,b,f,p,t,tp);
```

针对例 8.12,MATLAB 在神经网络工具箱中还给出了如下示范程序

```
ff = nnfgflag('DEMOBP: Learning with Momentum');
if (nargin > 0 & ~ff),error('Do not call DEMOBP5 with arguments.'),end
if (ff)
   X = get(gcf,'userdata');
   w = X(1) ;
   mc = X(2) ;
   h = X(3) ;
   mc _ slider = X(4) ;
```

```
    mc_value = X(5) ;
  end
  default_w = -0.9;
  default_mc = 0.98;
  b = 3;
  p = [-6.0 -6.1 -4.1 -4.0 +5.0 +5.1 +6.0 +6.1];
  t = [+0.0 +0.0 +0.97 +0.99 +0.01 +0.03 +1.0 +1.0];
  wmin = -2;
  wmax = 2;
  w_range = wmin:0.05:wmax;
  b_range = 3;
  me = 300;
  eg = 0;
  er = 1.04; er = 10;
  lr = 0.05;
  this = 'demobp5';
  if (nargin == 0 & ff)
  elseif (nargin == 0 & ff == 0)
    clc
    disp('INITFF-- Initializes a feed-forward network.')
    disp('TRAINBPM——Trains a feed-forward network with bp + momen-
tum.')
    disp('SIMFF——Simulates a feed-forward network.')
    disp(' ')
    disp('LEARNING WITH MOMENTUM:')
    disp(' ')
    disp('By adding momentum to backpropagation, network can often')
    disp('escape shallow error minima.')
    disp(' ')
    %   FIGURE
    clf reset
    colordef(gcf,'none')
    setfsize(400,310) ;
    set(gcf,...
      'numbertitle','off',...
      'name','DEMOBP5: Learning with Momentum',...
      'color',[0.701961 0.701961 0.701961],...
      'resize','off',...
      'colormap',hsv);
```

```
% AXIS
set(gca,...
  'box','on',...
  'units','pixels',...
  'position',[30 120 360 180],...
  'xlim',[min(w_range) max(w_range)],...
  'ylim',[2 4.5],...
  'xtick',[],...
  'ytick',[],...
  'ButtonDownFcn','disp(''AXIS'')',...
  'view',[0 90],...
  'color',[0 0.5 0.5])
xlabel('Weight')
ylabel('Sum-Squared Error')
hx = get(gca,'xlabel');
hy = get(gca,'ylabel');
set(hx,'color',[0 0 0])
set(hy,'color',[0 0 0])
% DRAW
es = errsurf(p,t,w_range,b,'logsig');
hold on
E = [es es * 0];
W = [w_range fliplr(w_range)];
fill(W,E,'r')
plot3(w_range,es,es * 0+0.1,'y','linewidth',1) ;
w = default_w;
SSE = sumsqr(t-logsig(w * p,b));
h = plot(w,SSE,'.g','markersize',40,'erasemode','xor');
% MIDDLE FRAME
middle_frame = uicontrol;
set(middle_frame,...
  'style','frame',...
  'position',[10 60 380 40])
% LOWER FRAME
middle_frame = uicontrol;
set(middle_frame,...
  'style','frame',...
  'position',[10 10 380 40])
drawnow
```

```
% GO BUTTON
go_button = uicontrol;
set(go_button,...
  'position',[20 20 60 20],...
  'string','Go',...
  'callback',nncallbk(this,'go'))
% SELECT BUTTON
select_button = uicontrol;
set(select_button,...
'position',[130 20 60 20],...
  'string','Select',...
  'callback',nncallbk(this,'select'))
% DEFAULT BUTTON
default_button = uicontrol;
set(default_button,...
  'position',[210 20 60 20],...
  'string','Default',...
  'callback',nncallbk(this,'default'))
% CLOSE BUTTON
close_button = uicontrol;
set(close_button,...
  'position',[320 20 60 20],...
  'string','Close',...
  'callback',nncallbk(this,'close'))
% MOMENTUM CONSTANT SLIDER
mc = default_mc;
mc_slider = uicontrol;
set(mc_slider,...
  'style','slider',...
  'position',[90 72 240 16],...
  'callback',nncallbk(this,'momentum'),...
  'value',mc)
% MOMENTUM CONSTANT TITLE
mc_title = uicontrol;
set(mc_title,...
  'style','text',...
  'string','Momentum Constant:',...
  'horizontalalignment','left',...
  'position',[20 62 70 30])
```

```
    %  MOMENTUM CONSTANT VALUE
    mc_value = uicontrol;
    set(mc_value,...
      'style','text',...
      'string',num2str(mc),...
      'horizontalalignment','left',...
      'position',[350 70 30 16])
%  GO
%  ==
elseif strcmp(lower(cmd),'go')
    %  PRESENTATION PHASE
    a = logsig(w * p,3)  ;
    e = t-a;
    sse = sumsqr(e);
    %  INITIALIZE MOMENTUM
    current_mc = 0; dw = 0;
    fprintf('\n Training...')
    colormap(hsv);
    delete(plot(w,sse,'.'));
    drawnow;
    for i=1:me
      %  CHECK PHASE
      if sse < eg, i=i-1; break, end
      %  LEARNING PHASE
      d = deltalog(a,e);
      dw = learnbpm(p,d,lr,current_mc,dw);
      if abs(dw) < 0.0001, i=i-1; break, end
      new_w = w + dw;
      %  PRESENTATION PHASE
      new_a = logsig(new_w * p,b);
      new_e = t-new_a;
      new_sse = sumsqr(new_e);
      %  MOMENTUM PHASE
      if new_sse > sse * er
        mc = 0;
      else
        if new_sse < sse, current_mc = mc; end
        w=new_w; a=new_a; e=new_e; sse=new_sse;
      end
```

```
        if (w<wmin) | (w>wmax), dw = -dw; end
        %   DISPLAY PROGRESS
        delete(h)
        h = plot(w,sse,'.g','markersize',40,'erasemode','xor');
        drawnow
    end
    fprintf('done. \n\n')
    delete(plot(w,sse,'.'));
%   DEFAULT
%   =======
elseif strcmp(lower(cmd),'default')
    w = -0.9;
    mc = default_mc;
    delete(h)
    sse = sumsqr(t-logsig(w * p,b));
    delete(plot(w,sse,'.'));
    h = plot(w,sse,'.g','markersize',40,'erasemode','xor');
    set(mc_slider,'value',mc);
    set(mc_value,'string',num2str(mc));
%   CLOSE
%   =====
elseif strcmp(lower(cmd),'close')
    close;
    return
%   MOMENTUM
%   ========
elseif strcmp(lower(cmd),'momentum')
    mc = get(mc_slider,'value');
    set(mc_value,'string',sprintf('%.2f',mc));
    drawnow;
%   SELECT
%   ======
elseif strcmp(lower(cmd),'select')
    delete(h)
    sse = sumsqr(t-logsig(w * p,b));
    delete(plot(w,sse,'.'));
    th = centext('* CLICK ON ME *');
    set(th,'color',[1 1 1]);
    set(gcf,'pointer','cross')
```

```
    [w,dummy] = ginput(1) ;
    set(gcf,'pointer','arrow')
    delete(th);
    sse = sumsqr(t-logsig(w * p,b));
    delete(plot(w,sse,'.'));
    h = plot(w,sse,'.g','markersize',40,'erasemode','xor');
end
%   STORE INTERFACE VARIABLES
%   ==========================
set(gcf,'userdata',[w mc h mc_slider mc_value_)
```

例 8.13　程序的运行结果的进一步解释如下：一维误差曲线如图 8.20 所示. 可以看到在误差曲线上有两个误差最小值，一个为局部极小值在左边，右边的为全局最小值. 如果在训练时，动量因子 mc 选取为 0，网络以纯梯度法进行训练，此举的结果如图 8.21 所示，其误差的变化趋势是以简单的方式“滚到”局部极小值的底部就再也停止不动了. 当采用附加动量法后，网络的训练则可以自动地避免陷入这个局部极小值. 这个结果如图8.22所示.

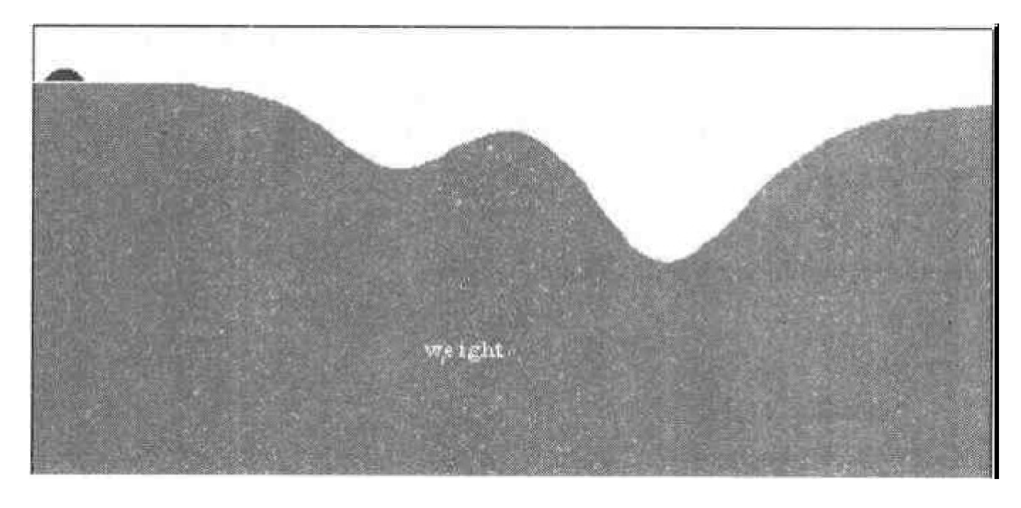


图 8.20　网络训练误差曲线

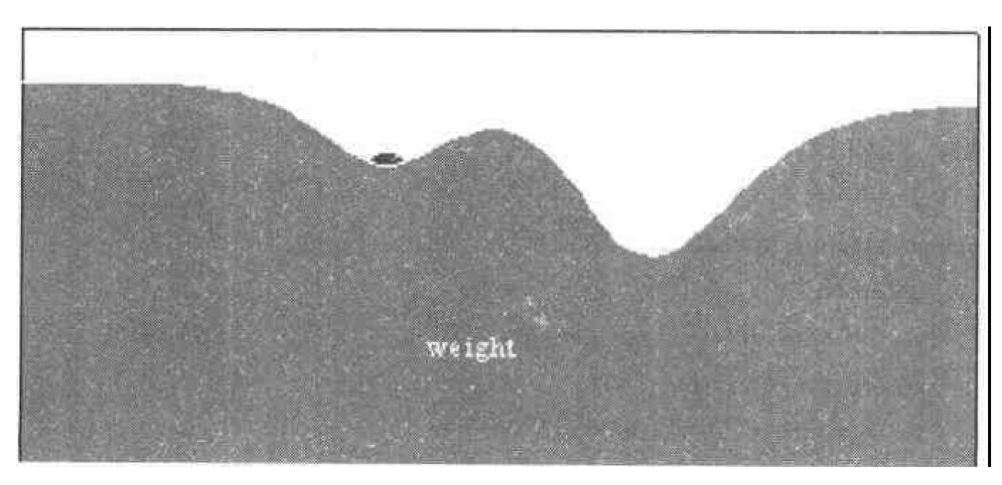


图 8.21　mc＝0 时的训练结果

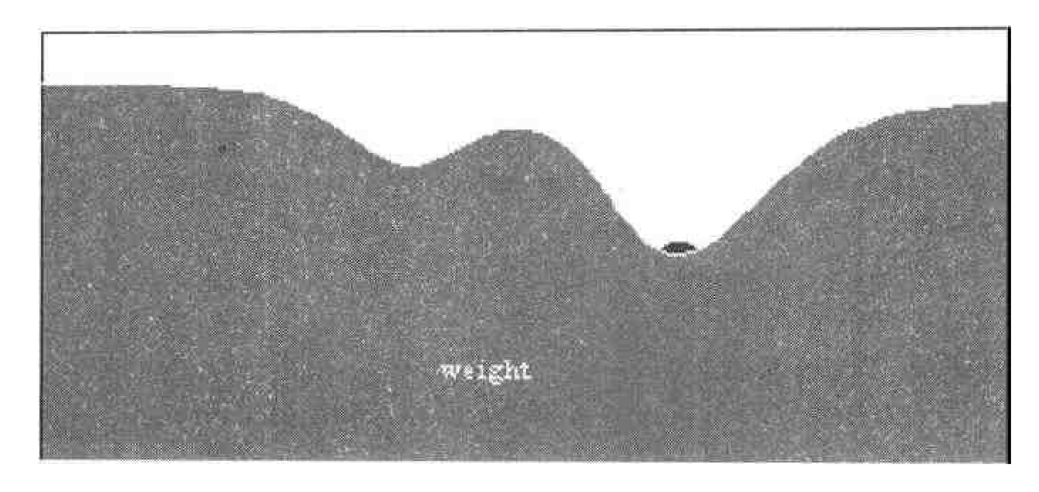


图 8.22　采用动量附加法的训练结果

例 8.14　自适应学习率调整算法来训练 BP 网络(在此仅给出本例的 MATLAB 程序).

```
clf reset
figure(gcf)
colordef(gcf,'none')
setfsize(500,200) ;
echo on
clc
%   INITFF——Initializes a feed-forware network.
%   TRAINBPX——Trains a feed-forward network with faster backpropagation.
%   SIMUFF——Simulates a feed-forward network.
%   FUNCTION APPROXIMATION WITH FASTER BACKPROPAGATION:
%   Using the above functions a two-layer network is trained
%   to respond to specific inputs with target outputs.
pause %   Strike any key to continue...
clc
%   DEFINING A VECTOR ASSOCATION PROBLEM
%   ==============================
%   P defines twenty-one 1-element input vectors (column vectors):
P = -1:.1:1;
%   T defines the associated 1-element targets (column vectors):
T = [-0.9602 -0.5770 -0.0729 0.3771 0.6405 0.6600 0.4609 ...
     0.1336 -0.2013 -0.4344 -0.5000 -0.3930 -0.1647 0.0988 ...
     0.3072 0.3960 0.3449 0.1816 -0.0312 -0.2189 -0.3201];
pause % Strike any key to see these data points...
```

```
clc
%   PLOTTING THE DATA POINTS
%   ==========================
%   Here the data points are plotted:
plot(P,T,'+');
title('Training Vectors');
xlabel('Input Vector P');
ylabel('Target Vector T');
%   The function the neural network learns must pass through
%   these data points.
pause % Strike any key to design the network...
clc
%   DESIGN THE NETWORK
%   ====================
%   A two-layer TANSIG/PURELIN network will be trained.
%   The number of hidden TANSIG neurons should reflect the
%   complexity of the problem.
S1 = 5;
%   INITFF is used to initialize the weights and biases for
%   the TANSIG/PURELIN network.
[w1,b1,w2,b2] = initff(P,S1,'tansig',T,'purelin');
echo off
k = pickic;
if k == 2
w1 = [3.5000; 3.5000; 3.5000; 3.5000; 3.5000];
b1 = [-2.8562; 1.0774; -0.5880; 1.4083; 2.8722];
w2 = [0.2622 -0.2375 -0.4525 0.2361 -0.1718];
b2 = [0.1326];
end
echo on
clc
%   TRAINING THE NETWORK
%   ====================
%   TRAINBP uses backpropagation to train feed-forward networks.
df = 10;      % Frequency of progress displays (in epochs).
me = 8000;    % Maximum number of epochs to train.
eg = 0.02;    % Sum-squared error goal.
lr = 0.01;    % Learning rate.
tp = [df me eg lr];
```

```
% Training begins...please wait (this takes a while!)...
[w1,b1,w2,b2,ep,tr] = trainbpx(w1,b1,'tansig',w2,b2,'purelin',P,T,
tp);
% ...and finally finishes.
% TRAINBP has returned new weight and bias values, the number
% of epochs trained EP, and a record of training errors TR.
pause % Strike any key to see a plot of errors...
clc
% PLOTTING THE ERROR CURVE
% ==========================
% Here the errors are plotted with respect to training epochs:
plottr(tr,eg);
pause % Strike any key to use the function approximator...
clc
% USING THE PATTERN ASSOCIATOR
% ================================
% We can now test the associator with one of the origonal
% inputs, 0.5, and see if it returns the target, 0.3960.
p = 0.5;
a = simuff(p,w1,b1,'tansig',w2,b2,'purelin')
% The result is fairly close. Training to a lower error
% goal would result in an even closer approximation.
echo off
disp('End of DEMOBP6')
```

例 8.15 运用 L-M 算法训练 BP 网络(在此仅给出本例的 MATLAB 程序).

```
clf reset
figure(gcf)
colordef(gcf,'none')
setfsize(500,200) ;
echo on
clc
% INITFF——Initializes a feed-forware network.
% TRAINLM——Trains a feed-forward network with faster backpropaga-
tion.
% SIMUFF——Simulates a feed-forward network.
% FUNCTION APPROXIMATION WITH LEVENBERG-MAR-
QUARDT:
% Using the above functions a two-layer network is trained
% to respond to specific inputs with target outputs.
```

```
pause % Strike any key to continue...
clc
%   DEFINING A VECTOR ASSOCATION PROBLEM
%   ====================================
%   P defines twenty-one 1-element input vectors (column vectors):
P = -1:.1:1;
%   T defines the associated 1-element targets (column vectors):
T = [-0.9602 -0.5770 -0.0729 0.3771 0.6405 0.6600 0.4609 ...
     0.1336 -0.2013 -0.4344 -0.5000 -0.3930 -0.1647 0.0988 ...
     0.3072 0.3960 0.3449 0.1816 -0.0312 -0.2189 -0.3201];
pause % Strike any key to see these data points...
clc
%   PLOTTING THE DATA POINTS
%   ========================
%   Here the data points are plotted:
plot(P,T,'+');
title('Training Vectors');
xlabel('Input Vector P');
ylabel('Target Vector T');
%   The function the neural network learns must pass through
%   these data points.
pause % Strike any key to design the network...
clc
%   DESIGN THE NETWORK
%   ==================
%   A two-layer TANSIG/PURELIN network will be trained.
%   The number of hidden TANSIG neurons should reflect the
%   complexity of the problem.
S1 = 5;
%   INITFF is used to initialize the weights and biases for
%   the TANSIG/PURELIN network.
[w1,b1,w2,b2] = initff(P,S1,'tansig',T,'purelin');
echo off
k = pickic;
if k == 2
w1 = [3.5000; 3.5000; 3.5000; 3.5000; 3.5000];
b1 = [-2.8562; 1.0774; -0.5880; 1.4083; 2.8722];
w2 = [0.2622 -0.2375 -0.4525 0.2361 -0.1718];
b2 = [0.1326];
```

```
end
echo on
clc
%    TRAINING THE NETWORK
%    ========================
%    TRAINLM uses Levenberg-Marquardt to train feed-forward networks.
df = 10;        % Frequency of progress displays (in epochs).
me = 8000;   % Maximum number of epochs to train.
eg = 0.02;    % Sum-squared error goal.
tp = [df me eg];
%    Training begins...please wait...
[w1,b1,w2,b2,ep,tr] = trainlm(w1,b1,'tansig',w2,b2,'purelin',P,T,tp);
%    ...and finally finishes.
%    TRAINLM has returned new weight and bias values, the number
%    of epochs trained EP, and a record of training errors TR.
pause % Strike any key to see a plot of errors...
clc
%    PLOTTING THE ERROR CURVE
%    ==========================
%    Here the errors are plotted with respect to training epochs:
plottr(tr,eg);
pause % Strike any key to use the function approximator...
clc
%    USING THE PATTERN ASSOCIATOR
%    ============================
%    We can now test the associator with one of the origonal
%    inputs, 0.5, and see if it returns the target, 0.3960.
p = 0.5;
a = simuff(p,w1,b1,'tansig',w2,b2,'purelin')
%    The result is off by about 7%. Training to a lower error
%    goal would result in a closer approximation.
echo off
disp('End of DEMOLM1')
```

第九章　径向基函数网络

9.1 引　　言

众所周知,BP网络用于函数逼近时,权值的调节采用的是负梯度下降法,这种调节权值的方法有它的局限性,即存在着收敛速度慢和局部极小等缺点.本章主要介绍在逼近能力、分类能力和学习速度等方面均优于BP网络的另一种网络——径向基函数网络.MATLAB的神经网络工具箱为径向基函数网络的仿真提供了丰富的函数,在MATLAB的工作空间中键入help radbasis,即可获得如下有关径向基函数网络的内容(本章所讨论的内容是在MATLAB5.3版本基础上进行的):

```
New networks.
      newrb      - Design a radial basis network.
      newrbe     - Design an exact radial basis network.
      newgrnn    - Design a generalized regression neural network.
      newpnn     - Design a probabilistic neural network.

Using networks.
      sim        - Simulate a neural network.

Weight functions.
      dist       - Euclidean distance weight function.
      dotprod    - Dot product weight function.
      normprod   - Normalized dot product weight function.

Net input functions.
      netprod    - Product net input function.
      netsum     - Sum net input function.

Transfer functions.
      compet     - Competitive transfer function.
      purelin    - Hard limit transfer function.
      radbas     - Radial basis transfer function.

Performance.
      mse        - Mean squared error performance function.
```

Signals.

ind2vec　- Convert indices to vectors.

vec2ind　- Convert vectors to indices.

Demonstrations.

demorb1　- Function approximation with a radial basis network.

demorb2　- Benchmarking function approximators.

demorb3　- Underlapping neurons.

demorb4　- Overlapping neurons.

demopnn1 - Probabilistic neural network classification.

demogrn1 - Generalized regression neural network approximation.

本章首先介绍径向基函数网络结构和理论，然后介绍 MATLAB 工具箱与径向基函数网络相关的函数及各种径向基网络的设计与仿真分析，最后给出径向基函数网络的设计和在状态观测器设计中的应用.

9.2　径向基函数神经网络

径向基函数(Radial Basis Function ，RBF)神经网络由三层组成，其结构如图 9.1 所示. 输入层节点只传递输入信号到隐层，隐层节点由像高斯函数那样的辐射状作用函数构成，而输出层节点通常是简单的线性函数.

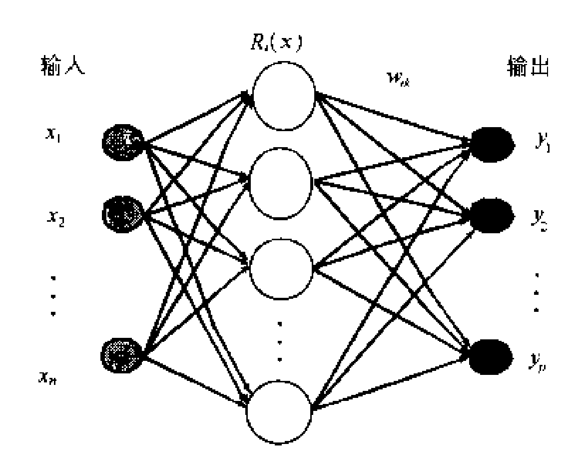


图 9.1　径向基函数神经网络

隐层节点中的作用函数(基函数)对输入信号将在局部产生响应，也就是说，当输入信号靠近基函数的中央范围时，隐层节点将产生较大的输出，由此看出这种网络具有局部逼近能力，所以径向基函数网络也称为局部感知场网络.

作为基函数的形式，有下列几种：

$$f(x)=\exp^{(x/\sigma)^2} \tag{9.1}$$

$$f(x)=\frac{1}{(\sigma^2+x^2)^{\alpha}}\quad,\quad \alpha>0 \tag{9.2}$$

$$f(x)=(\sigma^2+x^2)^{\beta} \quad , \quad \alpha<\beta<1 \tag{9.3}$$

上面这些函数都是径向对称的，但最常用的是高斯函数：

$$R_i(x)=\exp\left[-\frac{\| x-c_i \|^2}{2\sigma_i^2}\right] \quad , \quad i=1,2,\cdots,m \tag{9.4}$$

其中 x 是 n 维输入向量；c_i 是第 i 个基函数的中心，与 x 具有相同维数的向量，σ_i 是第 i 个感知的变量(可以自由选择的参数)，它决定了该基函数围绕中心点的宽度；m 是感知单元的个数. $\| x-c_i \|$ 是向量 $x-c_i$ 的范数，它通常表示 x 和 c_i 之间的距离，$R_i(x)$ 在 c_i 处有一个惟一的最大值，随着 $\| x-c_i \|$ 的增大，$R_i(x)$ 迅速衰减到零. 对于给定的输入 $x\in R^n$，只有一小部分靠近 x 的中心被激活.

从图 9.1 可以看出，输入层实现从 $x\rightarrow R_i(x)$ 的非线性映射，输出层实现从 $R_i(x)$ 到 y_k 的线性映射，即

$$y_k=\sum_{i=1}^{m} w_{ik}R_i(x) \quad , \quad k=1,2,\cdots,p \tag{9.5}$$

其中 p 是输出节点数.

其连接权的学习修正仍可采用 BP 算法. 由于 $R_i(x)$ 为高斯函数，因而对任意 x 均有 $R_i(x)>0$，从而失去局部调整权值的优点，而事实上，当 x 远离 c_i 时，$R_i(x)$ 已非常小，因此可作为 0 对待. 因此实际上只当 $R_i(x)$ 大于某一数值(例如 0.05) 时才对相应的权值 w_{ij} 进行修改. 经这样处理后 RBF 网络也同样具备局部逼近网络学习收敛快的优点. 同时这样近似处理，可在一定程度上克服高斯基函数不具备紧密性的缺点.

上述采用的高斯基函数，具备如下优点：

1) 表示形式简单，即使对于多变量输入也不增加太多的复杂性；

2) 径向对称；

3) 光滑性好，任意阶导数均存在；

4) 由于该基函数表示简单且解析性好，因而便于进行理论分析.

考虑到提高网络精度和减少隐层节点数，也可以将网络基函数改成多变量正态密度函数

$$R_i(x)=\exp\left(-\frac{1}{2}(x-c_i)K(x-c_i)^T\right) \tag{9.6}$$

其中 $K=E[(x-c_i)^T(x-c_i)]^{-1}$ 是输入协方差阵的逆. 注意这时的基函数式(9.6) 已不再是径向对称.

从理论上而言，RBF 网络和 BP 网络一样可近似任何的连续非线性函数. 两者的主要差别在于各使用不同的作用函数，BP 网络中的隐层节点使用的是 Sigmoid 函数，其函数值在输入空间中无限大的范围内为非零值，而 RBF 网络的作用函数则是局部的.

9.3 径向基函数神经网络的工具箱函数

9.3.1 网络设计函数

(1) newrb 设计一个径向基网络

格式：net = newrb(P,T,GOAL,SPREAD)

说明：用径向基函数网络逼近函数时，newrb 可自动增加径向基网络的隐层神经元，

直到均方误差满足为止.

其中

P　输入矢量.

T　目标矢量.

GOAL　均方误差，缺省时的值= 0.0.

SPREAD　径向基函数的分布，缺省时的值= 1.0.

例 9.1　已知输入和目标矢量：

```
P = [1 2 3];
T = [2.0 4.1 5.9];
```

设计一径向基网络(建议读者自己在 MATLAB 工作平台上运行,并分析结果)：

```
net = newrb(P,T);
```

然后，在网络的输入端输入一新的值：

```
P = 1.5;
```

运用仿真函数 sim：

```
Y = sim(net,P)
Y =
    2.6755
```

(2) newrbe 设计严格的径向基网络

格式:net = newrbe(P,T, SPREAD)

说明：用径向基函数网络逼近函数,newrbe 可迅速地设计一径向基网络，且在设计中误差为 0.

其中

P　输入矢量.

T　目标矢量.

SPREAD　径向基函数的分布，缺省时的值= 1.0.

例 9.2　已知输入和目标矢量：

```
P = [1 2 3];
T = [2.0 4.1 5.9];
```

设计一径向基网络(建议读者自己在 MATLAB 工作平台上运行,并分析结果)：

```
net = newrbe(P,T);
```

然后，在网络的输入端输入一新的值：

```
P = 1.5;
```

运用仿真函数 sim：

```
Y = sim(net,P)
Y =
    2.6755
```

(3) newgrnn 设计广义回归神经网络

格式:net = newgrnn(P,T,SPREAD)

说明：广义回归神经网络(GRNNs)是一种经常被应用在函数逼近中的径向基网络.

newgrnn 函数可迅速地设计 GRNNs 网络.

其中

P 输入矢量.

T 目标矢量.

SPREAD 径向基函数的分布，缺省时的值= 1.0.

例 9.3 已知输入和目标矢量：

```
P = [1 2 3];
T = [2.0 4.1 5.9];
```

设计一径向基网络(建议读者自己在 MATLAB 工作平台上运行,并分析结果)：

```
net = newgrnn(P,T);
```

然后,在网络的输入端输入一新的值：

```
P = 1.5;
```

运用仿真函数 sim：

```
Y = sim(net,P)
Y =
    3.3667
```

(4) newpnn 设计概率神经网络

格式:net = newpnn(P,T,SPREAD)

说明:概率神经网络是一种适合于模式分类的径向基网络.

其中

P 输入矢量.

T 目标矢量.

SPREAD 径向基函数的分布，缺省时的值= 1.0.

例 9.4 考虑模式分类问题,定义输入集：

```
P = [1 2 3 4 5 6 7];
```

类别指示集：

```
Tc = [1 2 3 2 2 3 1];
```

下面运用分类指示器转换成目标矢量,然后设计概率网络来进行验证. 运用 newpnn 函数

```
T = ind2vec(Tc)
T =
      (1,1)      1
      (2,2)      1
      (3,3)      1
      (2,4)      1
      (2,5)      1
      (3,6)      1
      (1,7)      1
net = newpnn(P,T);
```

```
Y = sim(net,P)
Y =
        (1,1)        1
        (2,2)        1
        (3,3)        1
        (2,4)        1
        (2,5)        1
        (3,6)        1
        (1,7)        1
Yc = vec2ind(Y)
Yc =
     1    2    3    2    2    3    1
```

9.3.2 权函数

(1) dist Euclidean 距离权函数

格式:Z = dist(W,P)

df = dist('deriv')

D = dist(pos)

说明: dist 是 Euclidean 距离权函数. 所谓权函数是给定输入而会得到相应的权输入. 两矢量 X 和 Y 之间的 Euclidean 距离 D 为

D = sum((x−y).^2) .^0.5

其中

W 为权阵.

P 为输入矢量.

dist('deriv') 返回'' ,因为 dist 没有导数函数.

pos 为神经元位置阵.

例 9.5 定义一随机着阵 W 和输入矢量 P,计算相应的权 Z.

```
W= rand(4,3) ;
P= rand(3,1) ;
Z= dist(W,P)
Z=
    0.6637
    0.7414
    0.6095
    1.0426
```

例 9.6 定义三维空间上排列 10 个神经元的一个位置随机阵,求它们的距离.

```
pos = rand(3,10) ;
D = dist(pos)
```

(请读者自己在 MATLAB 平台上运行此程序,并体会运行结果.)

(2) dotprod 点积权函数

格式:Z = dotprod(W,P)

df = dotprod('deriv')

说明: dotprod 是点积权函数. 所谓权函数是给定输入而会得到相应的权输入.

其中

W 为权阵.

P 为输入矢量.

例 9.7 定义随机阵 W 和输入矢量 P,计算相应的权 Z.

```
W = rand(4,3) ;
P = rand(3,1) ;
Z = dotprod(W,P)
Z =
    0.5302
    0.8485
    0.5704
    1.0927
```

(3) normprod 规范点积(Normalized dot produc)权函数

格式:Z = normprod(W,P)

df = normprod('deriv')

说明: dist 是 Euclidean 距离权函数. normprod 返回值按如下算法求得:

z = w * p/sum(p)

其中

W 为权阵.

P 为输入矢量.

normprod('deriv') 返回'' ,因为 normprod 没有导数函数.

例 9.8 定义一随机着阵 W 和输入矢量 P,计算相应的权 Z.

```
W = rand(4,3) ;
P = rand(3,1) ;
Z = normprod(W,P)
Z =
    0.5669
    0.7470
    0.6486
    0.4614
```

9.3.3 网络输入函数

(1) netprod 网络输入的积函数

格式:N = netprod(Z1,Z2,...)

df = netprod('deriv')

说明：netprod 是网络输入的积函数，网络输入函数结合它的加权输入和阈值计算一层的网络输出.

其中

Zi S×Q 维矩阵.

netprod('deriv') 返回 netprod 的导数函数.

例 9.9 用计算两组加权输入矢量.

```
z1 = [1 2 4;3 4 1];
z2 = [-1 2 2; -5 -6 1];
n = netprod(z1,z2)
n =
     -1      4     8
     -15    -24    1
```

当加权输入具有相同阈值 b = [0; -1]，为了与 z1,z2 维数匹配，下面计算必须使用 concur 函数：

```
n = netprod(z1,z2,concur(b,3) )
n =
     0      0      0
     15     24     -1
```

(2) netsum 网络输入的和函数

格式：N = netsum(Z1,Z2,...)

df = netsum('deriv')

说明：netprod 是网络输入的和函数，网络输入函数结合它的加权输入和阈值计算一层的网络输出.

其中

Zi 为 S×Q 维矩阵.

netsum('deriv') 返回 netsum 的导数函数.

例 9.10 用计算两组加权输入矢量.

```
z1 = [1 2 4;3 4 1];
z2 = [-1 2 2; -5 -6 1];
n = netsum(z1,z2)
n =
     0      4     6
     -2    -2     2
```

当加权输入具有相同阈值 b = [0; -1]，为了与 z1,z2 维数匹配，下面计算必须使用 concur 函数：

```
n = netsum(z1,z2,concur(b,3) )
n =
     0      4     6
     -3    -3     1
```

9.3.4 radbas(径向基)传递函数

格式:A = radbas(N)

info = radbas(code)

说明:radbas 是径向基传递函数,此函数可由它的网络输入计算一层的输出.

其中

N 网络输入矢量阵.

radbas(code)函数可返回如下一些有用的信息:

'deriv' 返回导数函数名称.

'name' 返回全名.

'output' 返回输出范围.

'active' 返回激活输入范围.

例 9.11 运用 plot 命令观察如下 radbas 传递函数的运行结果,如图 9.2 所示.

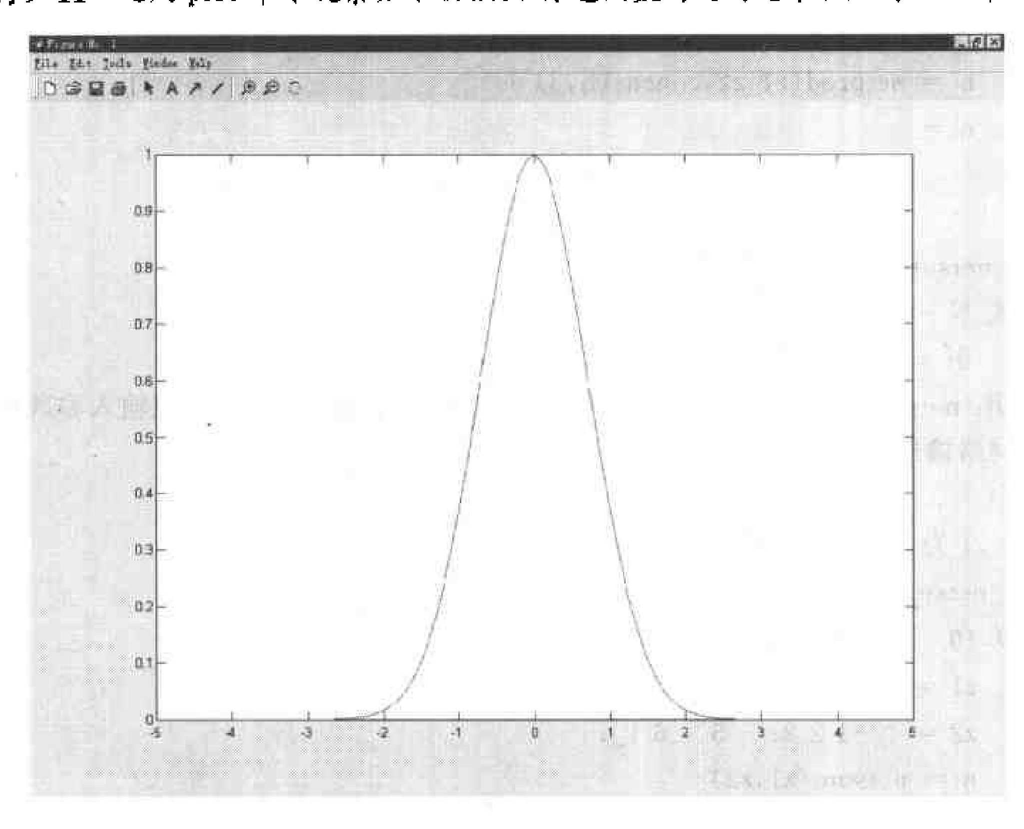


图 9.2 范例 9.11 的运行结果

```
n = -5:0.1:5;
a = radbas(n);
plot(n,a)
```

9.3.5 mse 均方误差性能函数

格式:perf = mse(e,x,pp)

perf = mse(e,net,pp)

```
info = mse(code)
```

说明：mse 是网络性能函数，此函数可按照均方误差测量网络性能.

其中

e　为误差矢量.

x　为所有权值和阈值组成的矢量.

pp　为性能参数.

net　为所有权值和阈值能组成矢量 x 的神经网络.

mse(code)函数可返回如下一些有用的信息：

'deriv'　为返回导数函数名称.

'name'　为返回全名.

'pnames'　为返回训练参数名.

'pdefaults'　为返回缺少的训练参数名.

例 9.12　创建一两层前馈网络(feed-forward network)，网络仅有一个输入值(−10,10)，隐层有四个 TANSIG 型神经元，一个 PURELIN 型输出神经元.

```
net = newff([-10 10],[4 1],{'tansig','purelin'});
```

再给定一组输入 p：

```
p = [-10 -5 0 5 10];
```

用网络的期望输出 t 减去实际输出 y 计算输出误差，然后再用 mse 函数对其性能进行分析：

```
p = [-10 -5 0 5 10];
t = [0 0 1 1 1];
y = sim(net,p)
y =
    0.4714     1.4040     1.2399     0.3581     0.9951
e = t-y
e =
    -0.4714    -1.4040    -0.2399     0.6419     0.0049
perf = mse(e)
perf =
    0.5326
```

9.3.6　变换函数

(1) ind2vec 将下标变换成单值矢量组

格式：vec = ind2vec(x)

说明：ind2vec 函数输入为包含 n 个下标的行矢量 x(<=1)，调用后可得到 m 行 n 列的矢量组矩阵，结果是矩阵中的每个矢量 i，除了由 x 中的第 i 个元素指定的位置为 1 外，其余元素为 0，结果矩阵的行数 m 等于 x 中最大的下标值.

例 9.13　定义一下标行矢量，利用 ind2vec 函数将其转换成单值矢量.

```
ind = [1 3 2 3]
```

```
vec = ind2vec(ind)
vec =
      (1,1)        1
      (3,2)        1
      (2,3)        1
      (3,4)        1
```

得到的矩阵为稀疏矩阵表示形式,利用 full 函数可得到完全存贮形式的矩阵.

```
y=full(vec)
y =
    1      0      0      0
    0      0      1      0
    0      1      0      1
```

(2) vec2ind 将单值矢量组变换成下标矢量

格式:ind = vec2ind(x)

说明:vec2ind 和 ind2vec 互为逆变换,vec2ind 函数输入为一个 m 行 n 列的矢量矩阵 x,调用后可得到 n 个下标值大小等于 0 的行矢量. x 中的每个矢量 i 除包含一个 1 外,其余均为 0,得到的行矢量包括这些非零元素的下标.

例 9.14 定义一矩阵,利用 vec2ind 函数将其转换成下标矢量.

```
vec = [1 0 0 0; 0 0 1 0; 0 1 0 1]
ind = vec2ind(vec)
ind =
     1      3      2      3
```

9.4 径向基函数网络的设计与应用

9.4.1 径向基函数网络的设计及在函数逼近上的应用

利用径向基函数网络来完成函数逼近任务. 输入样本和目标(期望)输出矢量分别为

```
P=-1:.1:1;
T=[-0.9602  -0.5770  -0.0729  0.3771  0.6405  0.6600
   0.4609  0.1336  -0.2013  -0.4344  0.5000  -0.3930
   -0.1647  0.0988  0.3072  0.3960  0.3449  0.1816
   -0.0312 -0.2189 -0.3201];
```

下面给出程序清单,图 9.3 至图 9.8 为程序的运行结果.

```
function slide=demorb1
%这是一个运用 playshow.m 和 makeshow.m 建立出的 slideshow 文件.
if nargout<1,
playshow demorb1
else
```

```
%画出第一幅图(Slide 1) :主菜单人机交互界面.
  slide(1) . code={
  'slideData=nnslides("start",slideData,"Function Approximation with Radi-
al Basis Networks");' };
  slide(1) . text={
    'NEWRB  -Creates a radial basis network.',
  'SIM-  Simulates a neural network.',
    'This demo uses the NEWRB function to create a radial basis network
that approximates a function defined by a set of data points.'};
%画出第二幅图(Slide 2) :待逼近的函数.
  slide(2) . code={
    'slideData=nnslides("axes",slideData);',
    'P = -1:.1:1;',
    'T = [-0.9602 -0.5770 -0.0729 0.3771 0.6405 0.6600 0.4609
0.1336 -0.2013 -0.4344 -0.5000 -0.3930 -0.1647 0.0988 0.3072 0.3960
0.3449 0.1816 -0.0312 -0.2189 -0.3201];'.
    'plot(P,T,"+");',
    'title("Training Vectors");',
    ' xlabel("Input Vector P");',
    ' ylabel("Target Vector T");' };
  slide(2) . text={
    'Define 21 inputs P and associated targets T:',
    '>> P = -1:.1:1;',
    '>> T = [-0.9602 -0.5770 -0.0729 0.3771 0.6405 0.6600
0.4609...',
    ' 0.1336 -0.2013 -0.4344 -0.5000 -0.3930 -0.1647 0.0988 ...',
    ' 0.3072 0.3960 0.3449 0.1816 -0.0312 -0.2189 -0.3201];',
    'Plot these 21 data points:',
    '>> plot(P,T,"+");'};
%画出第三幅图(Slide 3) :径向基函数图形.
  slide(3) . code={
    'p = -3:.1:3;',
    'a = radbas(p);',
    'plot(p,a)',
    'title("Radial Basis Transfer Function");',
    'xlabel("Input p");',
    'ylabel("Output a");' };
  slide(3) . text={
    'We would like to find a function which fits the 21 data points. One way
```

```
to do this is with a radial basis network. A radial basis network is a network with
two layers. A hidden layer of radial basis neurons and an output layer of linear neu-
rons. Here is the radial basis transfer function used by the hidden layer:',
        '>> p = -3:.1:3;',
        '>> a = radbas(p);',
        '>> plot(p,a)'};
    %画出第四幅图(Slide 4):径向基传递函数的加权和.
      slide(4).code={
        'slideData=nnslides("axes",slideData);',
        'a2 = radbas(p-1.5) ;',
        'a3 = radbas(p+2) ;',
        'a4 = a + a2*1 + a3*0.5;',
        'plot(p,a,"b--",p,a2,"b--",p,a3,"b--",p,a4,"m-")',
        'title("Weighted Sum of Radial Basis Transfer Functions");',
        'xlabel("Input p");',
        'ylabel("Output a");' };
      slide(4).text={
        'The weights and biases of each neuron in the hidden layer define the po-
sition and width of a radial basis function. Each linear output neuron forms a
weighted sum of these radial basis functions. With the correct weight and bias val-
ues for each layer, and enough hidden neurons, a radial basis network can fit any
function with any desired accuracy.',
         '>> plot(p,radbas(p)+radbas(p-1.5) +.05 * radbas(p+2) ,"m
-");',
        'Above is an example of three radial basis functions (in blue) are scaled
and summed to produce a function (in magenta).'};
    %画出第五幅图(Slide 5):由给定的p和t建立径向基网络,并设定训练误差和
学习速率.
      slide(5).code={
        'slideData=nnslides("blank",slideData);',
        'eg = 0.02; % sum-squared error goal.',
        'sc = 1; % spread constant radial basis functions.',
        'net=newrb(P,T,eg,sc);',
        '' };
      slide(5).text={
        'Th function NEWRB quickly creates a radial basis network which ap-
proximates the function defined by P and T.',
        'In addition to the training set and targets, NEWRB takes two argu-
ments:',
```

```
    '>> eg = 0.02; % sum-squared error goal',
    '>> sc = 1; % spread constant',
    '>> net=newrb(P,T,eg,sc);'};
%画出第六幅图(Slide 6):训练后的结果.
slide(6).code={
    'slideData=nnslides("axes",slideData);',
    'plot(P,T,"+");',
    'title("Training Vectors");',
    ' xlabel("Input Vector P");',
    ' ylabel("Target Vector T");',
    'X=-1:.01:1;',
    'Y=sim(net,X);',
    'hold on;',
    'plot(X,Y);',
    'hold off;',
    '' };
slide(6).text={
    'To see how the network performs, replot the training set:',
    '>> plot(P,T,"+");',
    'Simulate the network response for inputs over the same range:',
    '>> X=-1:.1:1;',
    '>> Y=sim(net,X);',
    'And plot the results on the same graph:',
    '>> hold on; plot(X,Y); hold off;'};
end
```

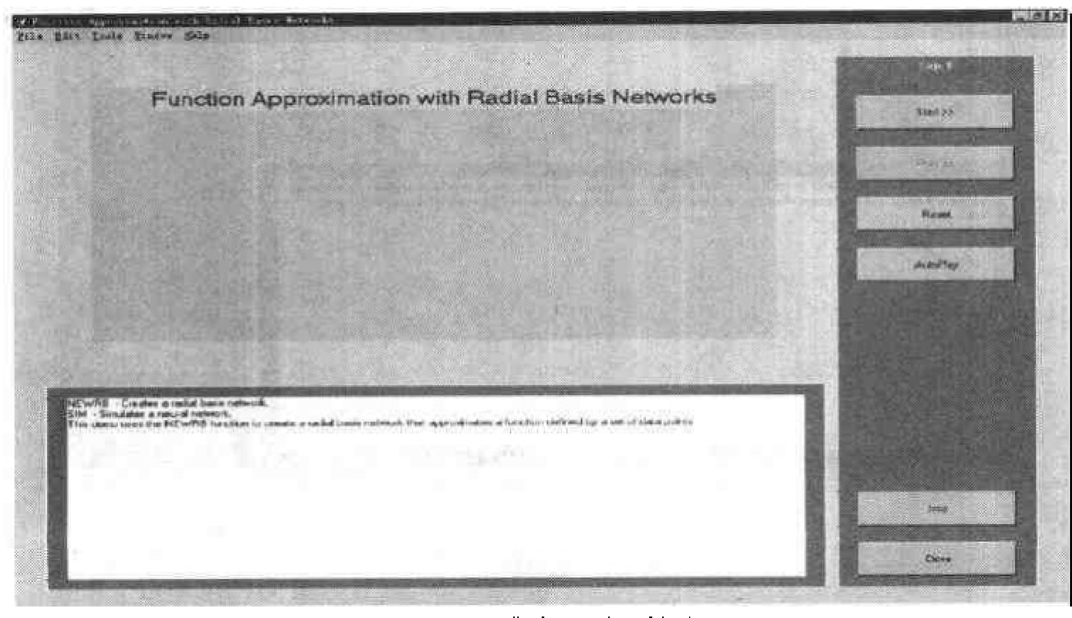


图 9.3 主菜单人机交互界面

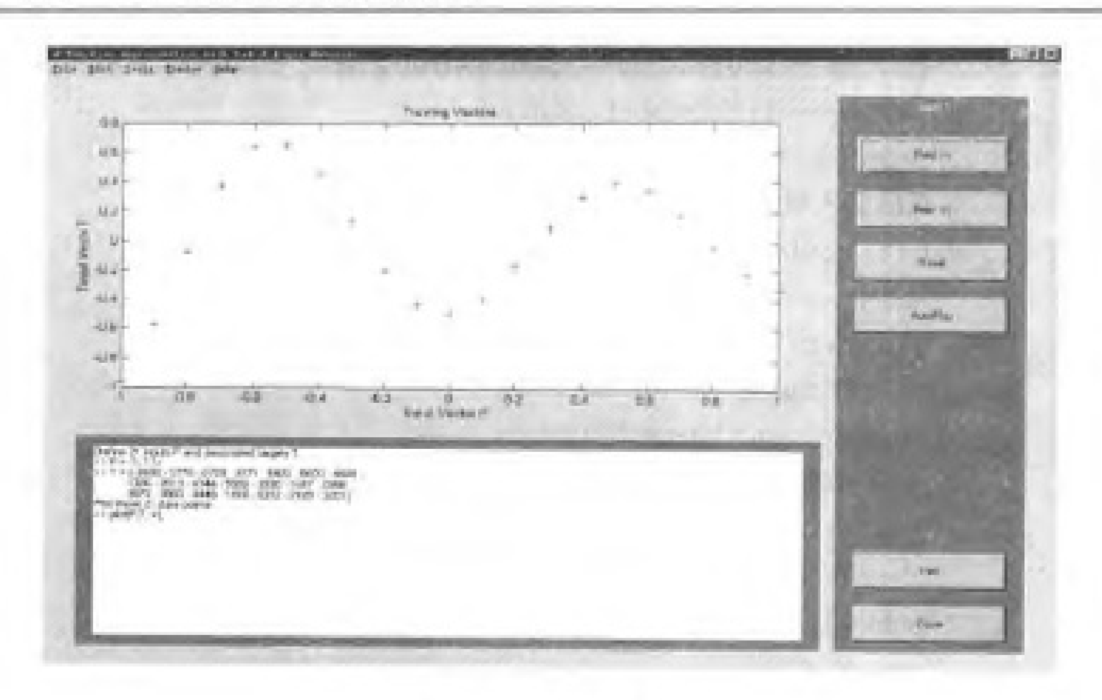


图 9.4　待逼近的函数

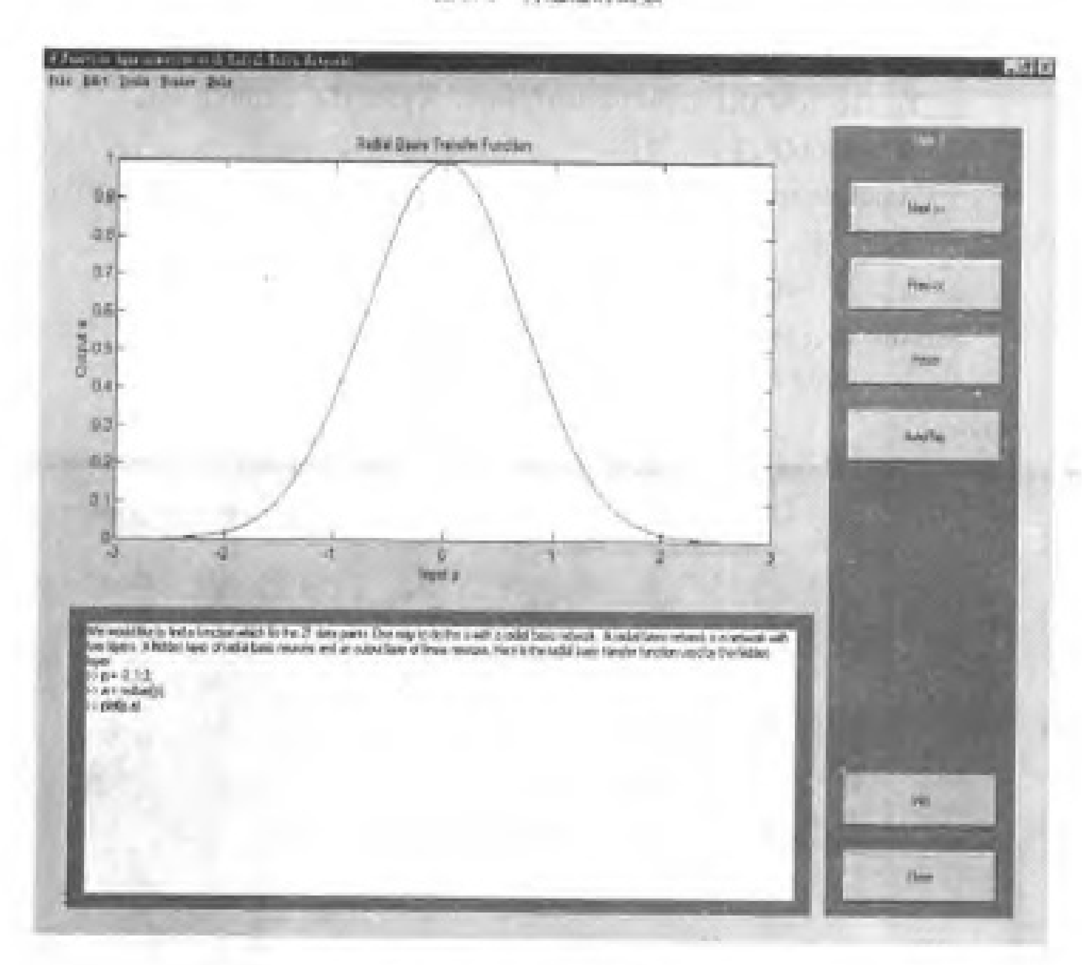


图 9.5　径向基函数图形

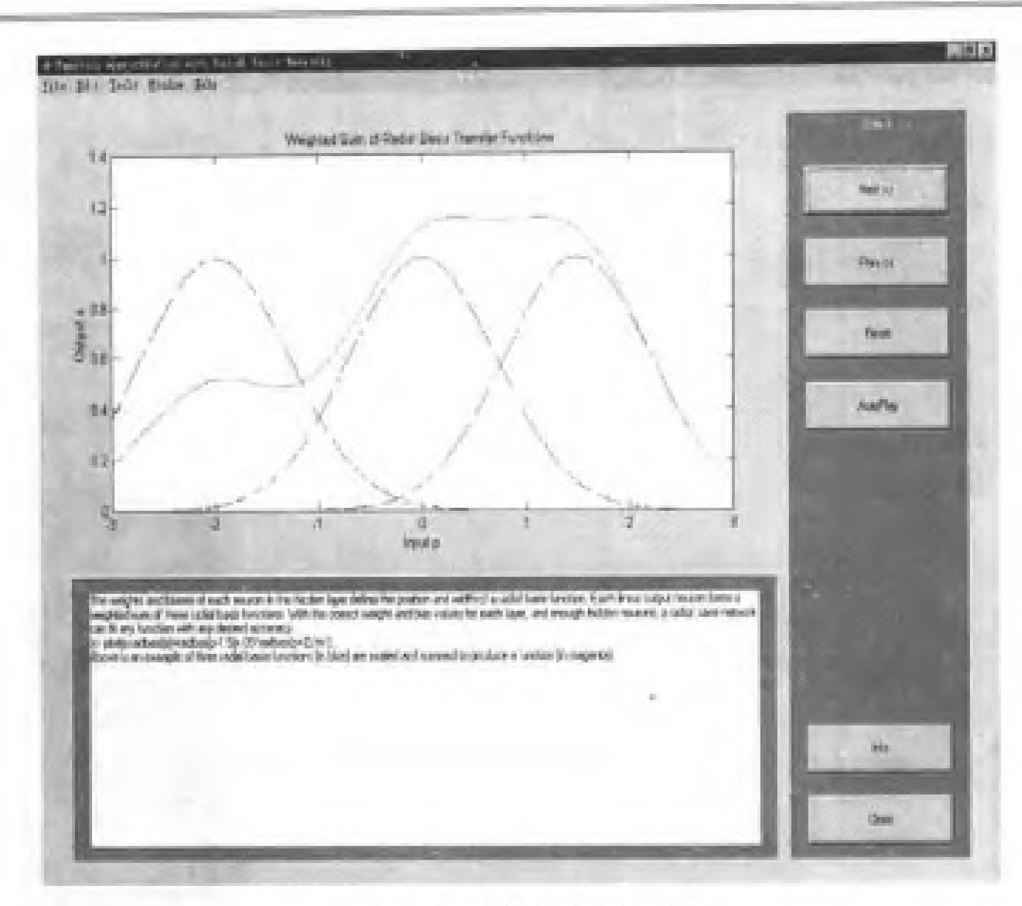


图 9.6 径向基传递函数的加权和

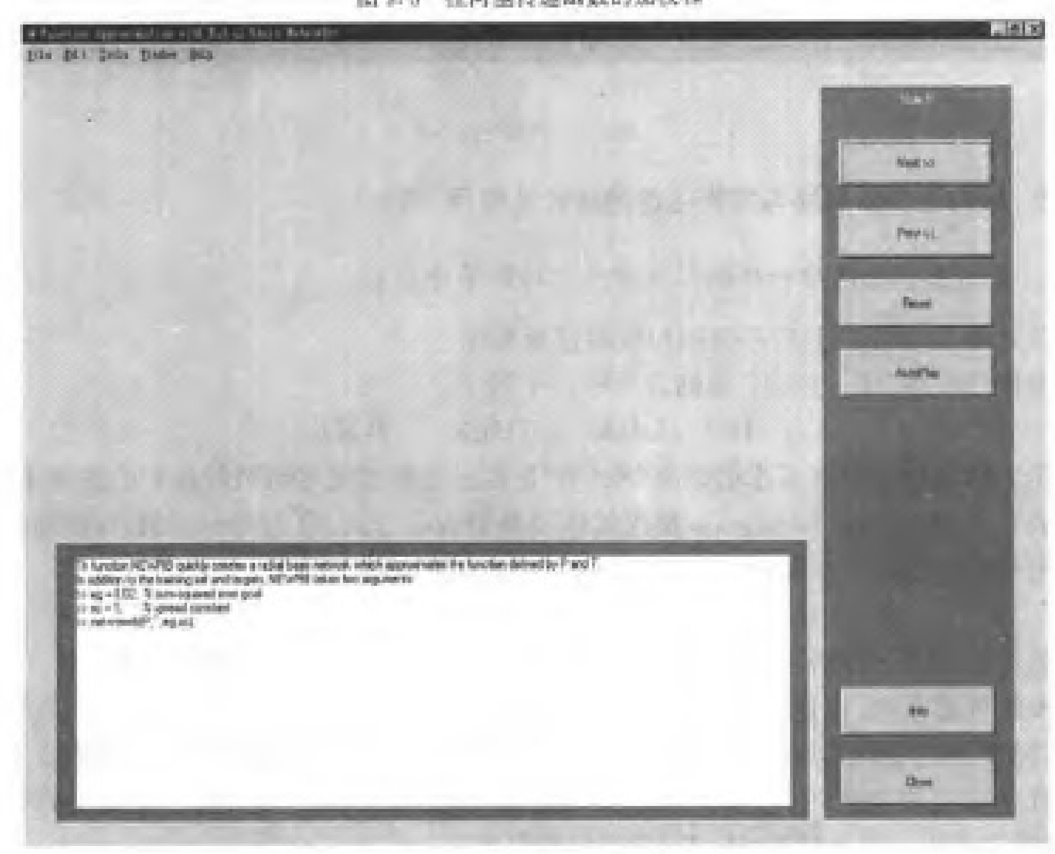


图 9.7 建立径向基网络

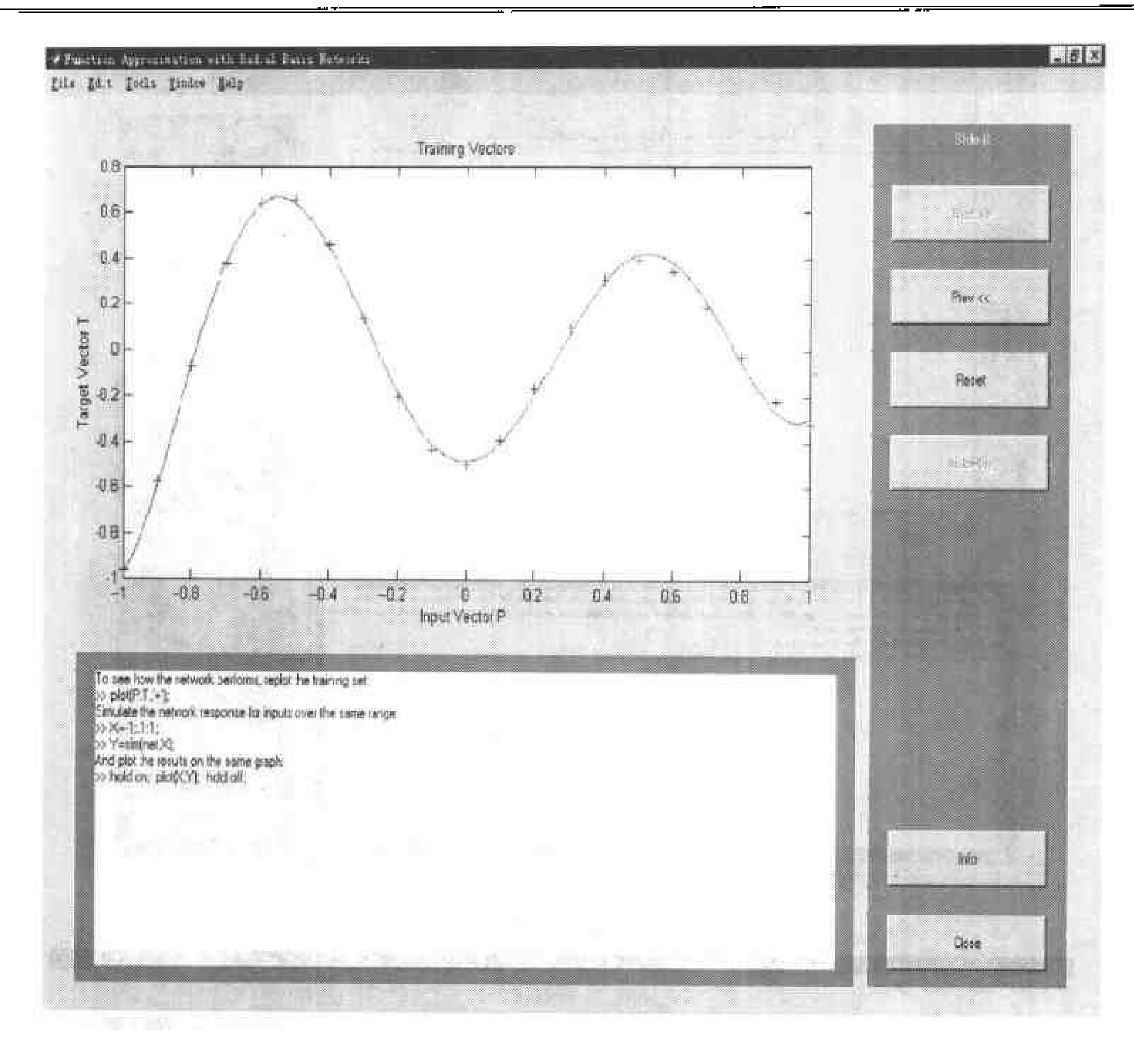


图 9.8 训练后的结果

9.4.2 径向基函数网络与模糊理论的结合及应用

1. 径向基函数网络和模糊推理系统的功能等价关系

(1) 模糊的“IF-THEN”规则和模糊推理系统

考虑用模糊“IF-THEN”规则表示的一个例子：

IF 压力高 THEN 容量小

这里压力和容量都是语言术语，“高”和“小”是由适当的隶属度函数特征化的语言术语（或语言表）. 采用Takagi 和 Sugeno 提出的模糊模型表示方式，给出另一个例子，模糊集仅包含在前件部分，例如，空气阻力 F 与运动物体的速度 V 的关系：

IF V 大 THEN $F=k*V^2$

这里前件的“大”是用语言表达的，后件部分是输入变量速度 V 的非模糊方程.

模糊推理系统，也是众所周知的模糊规则基系统、模糊模型、模糊联想记忆或模糊控制（在控制系统中应用时）. 模糊推理系统是由模糊“IF-THEN”规则的集合和语言表隶属度函数的数据库组成的，且推论机制称做模糊推理. 按 Takagi 和 Sugeno 提出的推理形式，假设规则库由两个模糊“IF-THEN”规则组成：

规则 1：IF x_1 is A_1 and x_2 is B_1 THEN $f_1=a_1x_1+b_1x_2+c_1$

规则 2：IF x_1 is A_2 and x_2 is B_2 　THEN $f_2=a_2x_1+b_2x_2+c_2$

则对其模糊推理过程可由图 9.9(a)给出，这里的第 i 个规则的起动强度可按前件部分隶属度值的 T-norm 获得

$$w_i=\mu_{A_i}(x_1)\mu_{B_i}(x_2) \quad 或 \quad w_i=\min\{\mu_{A_i}(x_1),\mu_{B_i}(x_2)\} \tag{9.7}$$

整个系统最后输出可以用每个规则的加权和表示

$$f(x)=\sum_{i=1}^{R}w_if_i(x) \tag{9.8}$$

或更常规地，用加权平均表示(如图 9.9(a)所示)

$$f(x)=\frac{\sum_{i=1}^{R}w_if_i(x)}{\sum_{i=1}^{R}w_i} \tag{9.9}$$

式中 R 是模糊"IF-THEN"规则的个数.

也可以直接转换模糊推理系统成为等价的自适应网络，如图 9.9(b)所示.

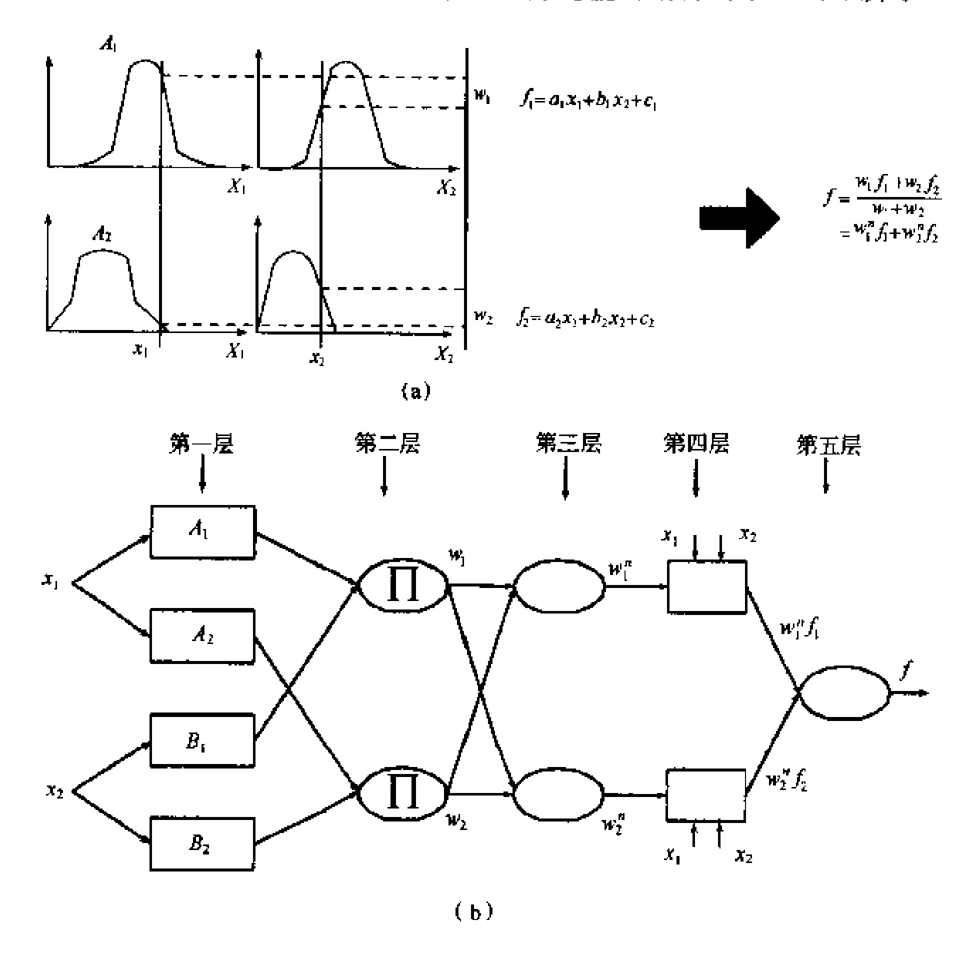


图 9.9　模糊规则的推理系统

(a) 模糊推理；(b) 自适应网络

(2) 功能等价和它的实现

从式(9.5)，式(9.7)，式(9.8)和式(9.9)中，显然可以看出如果下面条件成立，则

RBF 网络和模糊推理系统的功能是等价的：

1) 接收域单元的数等于模糊"IF-THEN"规则的数；

2) 每个模糊"IF-THEN"规则的输出是一个常值(即在图 9.9 中 a_1, b_1, a_2, b_2 都为 0)；

3) 每个规则的隶属度函数都选择具有与高斯函数相同形式；

4) 计算每个规则起动强度的 T-norm 算子是乘和；

5) RBF 网络和模糊推理系统使用相同的方法(加权平均或加权和)推导输出.

在这些条件下，语言表 A_1 和 B_1 的隶属度函数可以表示为

$$\mu_{A_1}(x_1) = \exp\left[-\frac{(x_1 - c_{A_1})^2}{\sigma_1^2}\right]$$

$$\mu_{B_1}(x_2) = \exp\left[-\frac{(x_2 - c_{B_1})^2}{\sigma_1^2}\right] \tag{9.10}$$

因此，规则 1 的起动强度为

$$\omega_1(x_1 \cdot x_2) = \mu_{A_1}(x_1)\mu_{B_1}(x_2) = \exp\left[-\frac{(\vec{x} - \vec{c}_1)^2}{\sigma_1^2}\right] = R_1(\vec{x}) \tag{9.11}$$

其中 $\vec{c}_1 = (c_{A_1} \cdot c_{B_1})$ 为相应接收域的中心.

类似地，ω_2 可以用同样的方法得出. 因此，在上面的条件下，RBF(具有两个接收域单元)与图 9.9(a)在功能上完全相同. 如果没有上面的限制，RBF 的功能仅仅是模糊推理系统的一个特例.

2. 基于自适应模糊系统(Adaptive Auzzy Systems , AFSs) 的径向基高斯函数网络 (Radial Basis Function ,RBF)

基于 AFSs 的 RBF 的基本特征是由前提和结论两部分构成的，且每一部分都包含有关可调整的参数集. 下面，给出三种基于 AFSs 的 RBF 网络类型，在这三种类型中，有一个共同点是在已知输入和被存贮前提事件之间，用和、积复合运算给出匹配关系：

$$R_i(x) = \exp\left[-\sum_{j=1}^{n} \frac{|x_j - c_{ji}|^2}{2\sigma_{ji}^2}\right] = \prod_j A_{ji}(x) \tag{9.12}$$

(1)类型 Ⅰ：结论是常值

假设有 m 个模糊规则，每个规则有 n 个输入和 p 个输出，第 i 个规则的形式如下：

Rule i: if(x_1 is A_{1i})and…and(x_n is A_{ni})
then(y_1 is a_{i1})and…and(y_p is a_{ip})

其中 a_{ik} 是常数.

由前面给出的等价推理结论，得其推理输出 y_k 为

$$y_k = \frac{\sum_{i=1}^{m} R_i(x) a_{ik}}{\sum_{i=1}^{m} R_i(x)} = \sum_{i=1}^{m} \hat{R}_i(x) \tag{9.13}$$

图 9.10 给出了基于 AFSs 的 RBF 网络(式(9.13) 的实现)，值得注意的是式(9.12) 的类

型属于类型 1,很容易由简单的 RBF 网络实现.

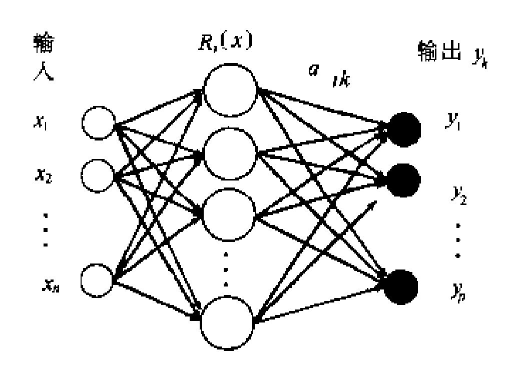


图 9.10 基于 AFSs 的 RBF 网络结构:类型 I

(2)类型 Ⅱ:后件是一阶线性方程

考虑 Sugen 模糊模型

$$\text{Rule } i: \begin{array}{l}\text{if}(x_1 \text{ is } A_{1i})\text{and}\cdots\text{and}(x_n \text{ is } A_{ni})\\ \text{then}(y_1 \text{ is } f_{i1})\text{and}\cdots\text{and}(y_p \text{ is } f_{ip})\end{array}$$

其中 $f_{ik}=a_{ik0}+a_{ik1}x_1+a_{ik2}x_2+\cdots+a_{ikn}x_n$.

由上节的结论,其模糊推理输出

$$\begin{aligned} y_k &= \frac{\sum_{i=1}^{m} R_i(x) f_{ik}}{\sum_{i=1}^{m} R_i(x)} = \sum_{i=1}^{m} \hat{R}_i(x) f_{ik} \\ &= \sum_{i=1}^{m} \hat{R}_i(x) a_{ik0} + \sum_{i=1}^{m} \hat{R}_i(x) (\sum_{l=1}^{n} a_{ikl} x_l) \end{aligned} \tag{9.14}$$

式(9.14) 可由图 9.11 所示的径向基函数网络实现.

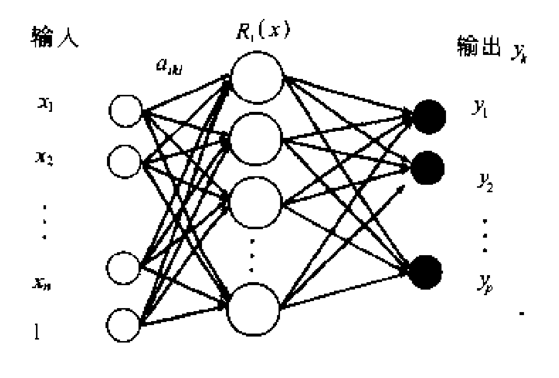


图 9.11 基于 AFSs 的 RBF 网络结构:类型 Ⅱ

(3) 类型 Ⅲ:后件是模糊变量

考虑模糊规则

$$\text{Rule } i: \begin{array}{l}\text{if}(x_1 \text{ is } A_{1i})\text{and}\cdots\text{and}(x_n \text{ is } A_{ni})\\ \text{then}(y_1 \text{ is } B_{i1})\text{and}\cdots\text{and}(y_p \text{ is } B_{ip})\end{array}$$

其中 B_{ik}是模糊集.

一般情况,模糊隶属度函数是正规凸函数,可由参数函数形式表示

$$B_{ik}=f(w_{ik},y_{ik}^*)$$

其中 w_{ik}是宽度向量，y_{ik}^* 是使 $\mu_{B_{ik}}(y_{ik}^*)=1$ 的元素. 如，含有三个隶属度函数的模糊集(图 9.12(a)所示)可表示为

$$B_{ik}=f(w_{ik1},w_{ik2},y_{ik}^*)$$

其中 $w_{ik1}=|y_{ik}^*-y_{ik}^l|$，$w_{ik2}=|y_{ik}^*-y_{ik}^r|$.

对于不规则形状的隶属度函数，可用五个参数惟一地表达(图 9.12(b))

$$B_{ik}=f(w_{ik1},w_{ik2},w_{ik3},w_{ik4},y_{ik}^*)$$

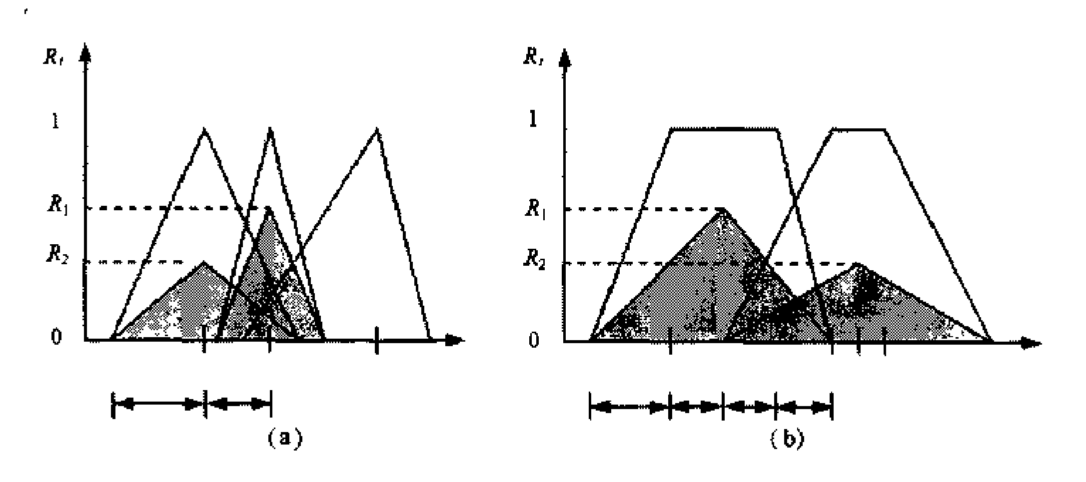


图 9.12　模糊数

(a) 三角形的模糊数；(b) 不规则几何形状的模糊数

模糊隶属度值可按下述方法计算

1) 等腰三角形的隶属度函数

$$\mu_i(w_{ik})=R_i(x)\frac{w_{ik}}{2}$$

其中 $w_{ik}=w_{ik1}+w_{ik2}$.

2) 一般三角形的隶属度函数

$$\mu_i(w_{ik})=R_i(x)\frac{w_{ik1}+w_{ik2}}{2}$$

3) 不规则几何形状的隶属度函数

$$\mu_i(w_{ik})=R_i(x)\frac{w_{ik1}+w_{ik2}+w_{ik3}+w_{ik4}}{2}$$

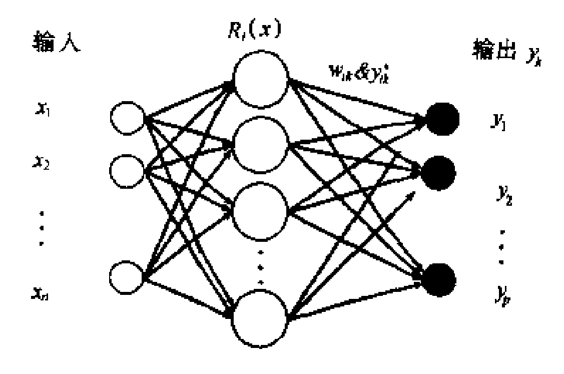


图 9.13　基于 AFSs 的 RBF 网络结构：类型 Ⅱ

应用重心法进行反模糊化计算，第 k 个元素的输出为

$$y_k=\frac{\sum_{i=1}^{m}\mu_i(w_{ik})y_{ik}^*}{\sum_{i=1}^{m}\mu_i(w_{ik})} \tag{9.15}$$

由上节，式(9.15) 可由图 9.13 所示的径向基函数网络实现.

3. 学习算法

基于模糊系统的径向基函数网络的学习过程是由确定最小知识规则数(隐层节点数)和调节隐层参数矢量所构成的，算法使网络从样本数据中估计出未知的规则，网络是否产生新的径向基节点，由有效半径的大小来确定，第 i 个隐层节点的有效半径 r_i 可用一个超球面 H 表达[5]

$$H(r_i)=\left\{x\left|d(x,c,\sigma)=\frac{(x-c)^2}{\sigma^2}\leqslant r_i^2\right.\right\} \tag{9.16}$$

(1) 递阶自组织学习算法

网络递阶算法学习过程如下：

第 1 步：设 $i=1,n=1$，i 和 n 分别代表隐层节点个数和第 n 个训练样本；

第 2 步：用第 n 个训练样本估计 $|y_n-t_n|$，这里 y_n 和 t_n 分别为第 n 个样本的网络输出值和期望值；

第 3 步：如果 $|y_n-t_n|>E$，E 为误差界，则转向第 4 步，否则转向第 7 步；

第 4 步：如果有一个隐层节点使训练样本落入在超球体 H 内，则转向第 5 步，否则，产生新的隐层节点，转向第 6 步. 新的隐层节点的参数按下述方法确定：

设　$i=i+1$；

均值向量 $c_i=$ 训练样本的输入；

标准偏差 $\sigma_i=\sigma_{init}$；

第 5 步：应用下节给出的修正方法调节参数向量；

第 6 步：转向第 2 步；

第 7 步：等待新的数据样本，然后转向第 1 步.

(2) 参数向量修正方法

基于训练模式，学习算法按照误差函数的副梯度下降方法，不断地更新网络参数. 第 n 个训练模式的误差参数 E_n，可定义为

$$E_n=\frac{1}{2}\sum_{k=1}^{p}(t_{nk}-y_{nk})^2 \tag{9.17}$$

其中 p 为输出单元的个数.

按照基于 AFSs 的 RBF 的结构，可定义网络参数向量：

基于 AFS1 的 RBF：$v_i=[c_{ji}^T,\sigma_{ji}^T,a_{ik}^T]^T$

基于 AFS2 的 RBF：$v_i=[c_{ji}^T,\sigma_{ji}^T,a_{ikj}^T]^T$

基于 AFS3 的 RBF：$v_i=[c_{ji}^T,\sigma_{ji}^T,y_{ik}^{*T},w_{ik}^T]^T$

参数更新修正规则

$$v_i^{\text{new}} = v_i^{\text{old}} + \eta \Delta v_i = v_i^{\text{old}} - \eta \frac{\partial E_n}{\partial v_i} \tag{9.18}$$

其中 η 为学习速率.

下面,对式(9.18)中的$\frac{\partial E_n}{\partial v_i}$,给出详细的推导,为了方便起见,下面推导中,省略标号中的下标 n.

1) 基于 AFSI 的 RBF:

在这种情况下,第 k 个元素的输出

$$y_k = \frac{\sum_{i=1}^{m} R_i(x) a_{ik}}{\sum_{i=1}^{m} R_i(x)} = \frac{u(c_{ji}, \sigma_{ji}, a_{ik})}{z(c_{ji}, \sigma_{ji})} \tag{9.19}$$

对于每个参数,E 的负梯度可进一步计算

$$\Delta c_{ji} = -\frac{\partial E}{\partial c_{ji}} = -\frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial R_i}\frac{\partial R_i}{\partial c_{ji}} = (t_k - y_k)\frac{a_{ik} - y_k}{z} R_i \frac{x_j - c_{ji}}{\sigma_{ji}^2} \tag{9.20}$$

$$\Delta \sigma_{ji} = -\frac{\partial E}{\partial \sigma_{ji}} = -\frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial R_i}\frac{\partial R_i}{\partial \sigma_{ji}} = (t_k - y_k)\frac{a_{ik} - y_k}{z} R_i \frac{(x_j - c_{ji})^2}{\sigma_{ji}^3} \tag{9.21}$$

$$\Delta a_i = -\frac{\partial E}{\partial a_{ij}} = -\frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial u}\frac{\partial u}{\partial a_{ik}} = (t_k - y_k)\frac{1}{z} R_i \tag{9.22}$$

2) 基于 AFS Ⅱ 的 RBF:

在这种情况下,第 k 个元素的输出

$$y_k = \frac{\sum_{i=1}^{m} R_i(x) f_{ik}}{\sum_{i=1}^{m} R_i(x)} = \frac{u(c_{ji}, \sigma_{ji}, a_{ikj})}{z(c_{ji}, \sigma_{ji})} \tag{9.23}$$

对于每个参数,E 的负梯度可进一步计算

$$\Delta c_{ji} = -\frac{\partial E}{\partial c_{ji}} = -\frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial R_i}\frac{\partial R_i}{\partial c_{ji}} = (t_k - y_k)\frac{a_{ik} - y_k}{z} R_i \frac{x_j - c_{ji}}{\sigma_{ji}^2} \tag{9.24}$$

$$\Delta \sigma_{ji} = -\frac{\partial E}{\partial \sigma_{ji}} = -\frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial R_i}\frac{\partial R_i}{\partial \sigma_{ji}} = (t_k - y_k)\frac{a_{ik} - y_k}{z} R_i \frac{(x_j - c_{ji})^2}{\sigma_{ji}^3} \tag{9.25}$$

$$\Delta a_i = -\frac{\partial E}{\partial a_{ij}} = -\frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial u}\frac{\partial u}{\partial f_{ik}}\frac{\partial f_{ik}}{\partial a_{ikj}} = (t_k - y_k)\frac{1}{2} R_i x_j \tag{9.26}$$

3) 基于 AFS Ⅲ 的 RBF:

在这种情况下,第 k 个元素的输出

$$y_k = \frac{\sum_{i=1}^{m} \mu_i(w_{ik}) y_{ik}^*}{\sum_{i=1}^{m} \mu_i(w_{ik})} = \frac{u(c_{ji}, \sigma_{ji}, y_{ik}^*, w_{ik})}{z(c_{ji}, \sigma_{ji}, w_{ik})} \tag{9.27}$$

其中 $\mu_i(w_{ik}) = R_i(x) w_{ik}/2$.

对于每个参数,E 的负梯度可进一步计算

$$\Delta c_{ji} = -\frac{\partial E}{\partial c_{ji}} = -\frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial R_i}\frac{\partial R_i}{\partial c_{ji}} = (t_k - y_k)\frac{y_{ik}^* - y_k}{z}\frac{w_{ik}}{2} R_i \frac{x_j - c_{ji}}{\sigma_{ji}^2} \tag{9.28}$$

$$\Delta\sigma_{ji} = -\frac{\partial E}{\partial\sigma_{ji}} = -\frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial R_i}\frac{\partial R_i}{\partial\sigma_{ji}} = (t_k - y_k)\frac{y_{ik}^* - y_k}{z}\frac{w_{ik}}{2}R_i\frac{(x_j - c_{ji})^2}{\sigma_{ji}^3} \tag{9.29}$$

$$\Delta w_{ik} = -\frac{\partial E}{\partial w_{ik}} = -\frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial\mu_i}\frac{\partial\mu_i}{\partial w_{ik}} = (t_k - y_k)\frac{y_{ik}^* - y_k}{z}\frac{R_i}{2} \tag{9.30}$$

$$\Delta y_{ik}^* = -\frac{\partial E}{\partial y_{ik}^*} = -\frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial u}\frac{\partial u}{\partial y_{ik}^*} = (t_k - y_k)\frac{1}{z}\mu_i \tag{9.31}$$

4. 非线性系统的故障诊断

(1) 自适应状态观测器设计

考虑非线性时变系统

$$\begin{aligned} x(k) &= f(x(k), u(k), \beta(k)) \\ y(k) &= g(x(k)) \end{aligned} \tag{9.32}$$

其中，$u\in E^l$；$y\in R^m$；$x\in R^n$；$f(\cdot)$为非线性函数；$g(\cdot)$为已知的非线性观测函数；$\beta(\cdot)$为系统随时间变化的参数，它是一个随时间慢变的非线性函数.

为了从非线性时变系统的输入$u(k)$和$y(k)$估计出系统的状态，用图 9.14 结构的基于模糊系统的径向基网络动态系统构成状态观测器，系统的输出作为估计器的一个输人.动态方程如下：

$$\begin{aligned} Z(k) &= r(Z(k), u(k), y(k)) \\ \hat{y}(k) &= g(Z(k)) \end{aligned} \tag{9.33}$$

其中$Z(k)\in R^n$，为基于模糊规则的径向网络动态系统的状态.

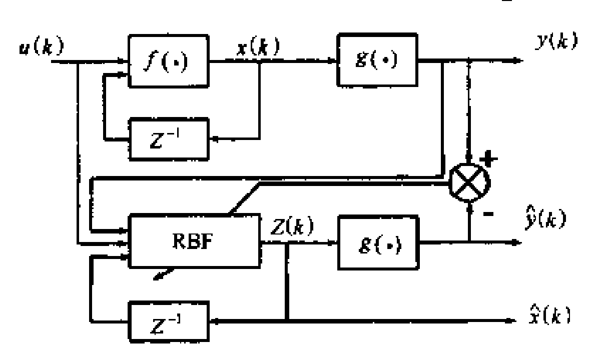


图 9.14 自适应状态观测器结构图

例 9.15 仿真系统

$$\begin{aligned} x(k+1) &= (1+\beta(k))\sin x(k) - 0.1x(k) + u(k) \\ y(k) &= 1.5x(k) + x(k-1) \\ \beta(k) &= \sin(0.035k) \end{aligned} \tag{9.34}$$

系统的输入采用白噪声，学习速率取为 0.2，按上节的算法分别用基于 AFSs Ⅰ，AFSs Ⅱ 和 AFSs Ⅲ 的 RBF 网络进行在线学习、估计，仿真观测器的规则节点分别稳定在 $m=20$、$m=17$ 和 $m=5$，图 9.15、图 9.16 和图 9.17 为仿真结果. 由仿真结果可以看出，系统初始收敛速度快（具体仿真程序这里不再给出）.

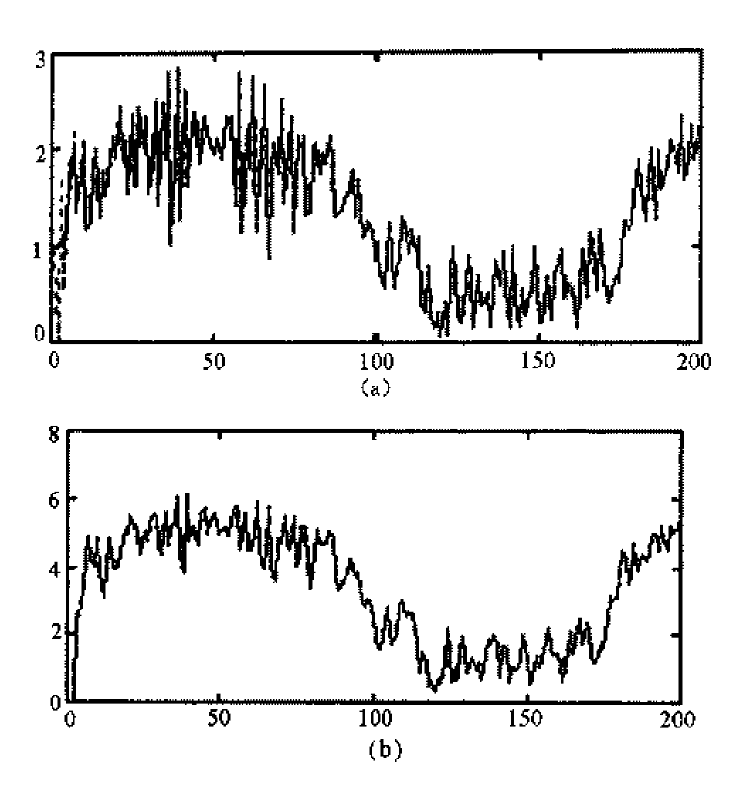


图 9.15 基于 AFSs Ⅰ 的 RBF 网络仿真结果

(a) 系统状态(实线)及观测器估计的状态(虚线)；(b) 系统输出(实线)及观测器估计的输出(虚线)

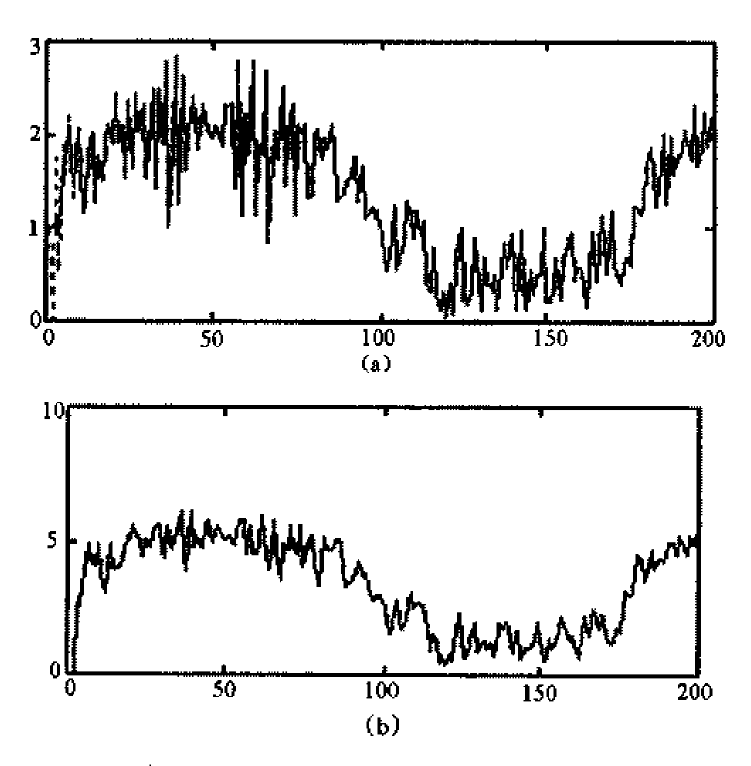


图 9.16 基于 AFSs Ⅱ 的 RBF 网络仿真结果

(a) 系统状态(实线)及观测器估计的状态(虚线)；(b) 系统输出(实线)及观测器估计的输出(虚线)

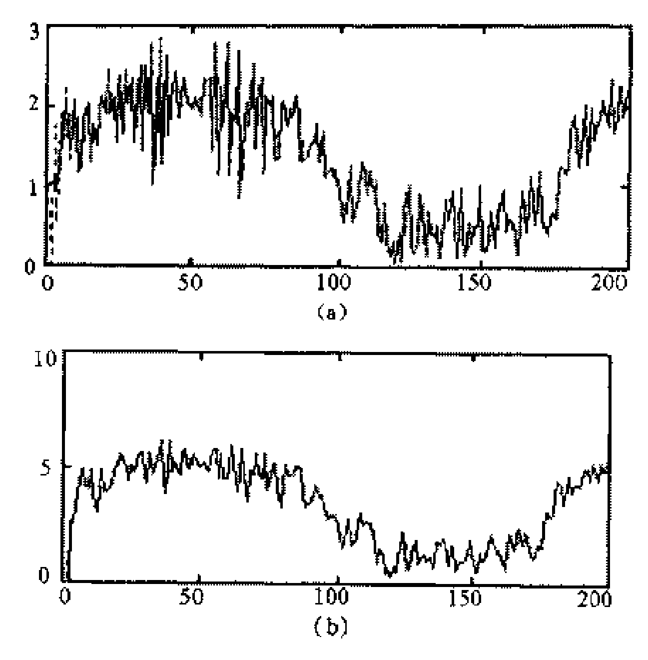


图 9.17 基于 AFSs Ⅱ 的 RBF 网络仿真结果

(a) 系统状态(实线)及观测器估计的状态(虚线);(b) 系统输出(实线)及观测器估计的输出(虚线)

(2) 非线性系统的故障诊断

对于式(9.32)系统,可以利用神经网络获得状态观测器式(9.33),利用这个状态观测器的输出值,进行下一步的系统的输出预报,便可以实现系统的故障检测,定义如下系统预报输出残差:

$$\varepsilon(k+1)=y(k+1)-\hat{y}(k+1) \tag{9.35}$$

其中 $\varepsilon(k)$ 为系统的预报输出残差,根据状态观测器设计的特点,$\varepsilon(k)$ 应很快衰减为 0,从而达到进行预报的目的. 但是,当系统存在故障时,相当于系统的物理结构发生了变化,即系统模型发生了变化,由于神经网络的自学习需要一个过程,所以,在这一时刻对状态的跟踪能力下降,导致系统的输出预报残差突变,利用这种突变就可以检测故障,设

$$\gamma(k)=\varepsilon(k)^T W\varepsilon(k) \tag{9.36}$$

其中 W 为对角加权阵,可根据实际问题的具体特征选取. 于是,故障检测规则为

$$\gamma(k)=\begin{cases}\leqslant T,\text{on} \quad \text{fault}\\ >T, \quad \text{fault}\end{cases} \tag{9.37}$$

式中 T 为故障检测的阈值.

考虑例 9.14,设系统在 $k=50$ 时出现故障,这里仅用基于 AFSs Ⅱ 的 RBF 网络,采用上述方法检测,当输入 $u(k)=1$ 时的仿真结果如图 9.18 所示,当输入 $u(k)$ 为随机数时的仿真结果如图 9.19 所示.

上述提出的自适应观测器设计方法,由于网络基函数形式简单,即使多变量输入也不增加太多的复杂性,所以很容易扩展到多输入多输出系统中,并且 FRBFs 也能同时处理定性和定量知识,有利于实际应用.

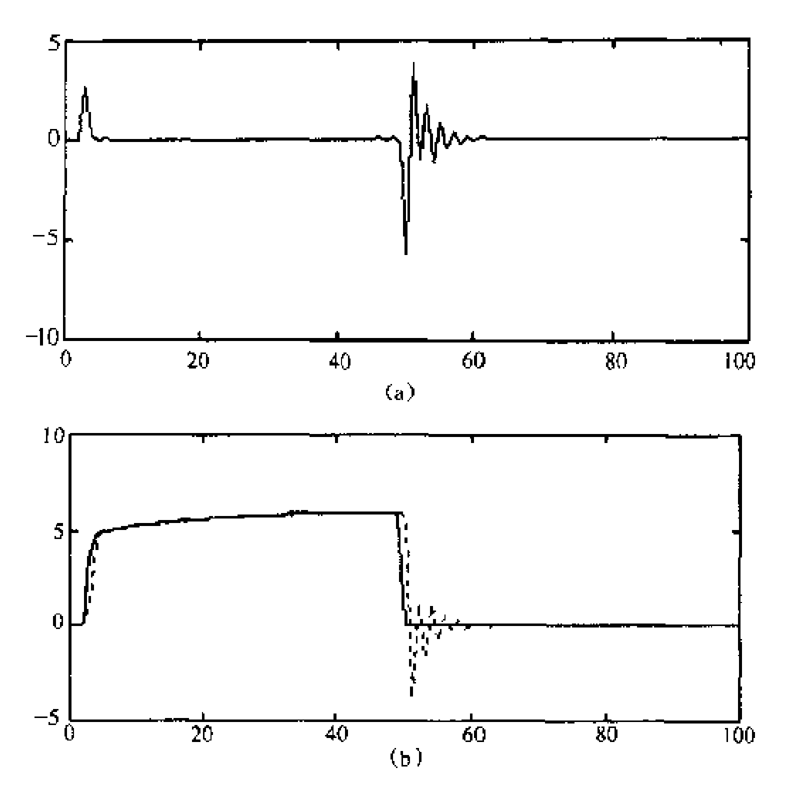


图 9.18　系统在 $k=50$ 时发生故障时的仿真结果

(a)为系统故障时的输出预报残差曲线；(b)系统故障时的实际输出(实线)和预报输出曲线(虚线)

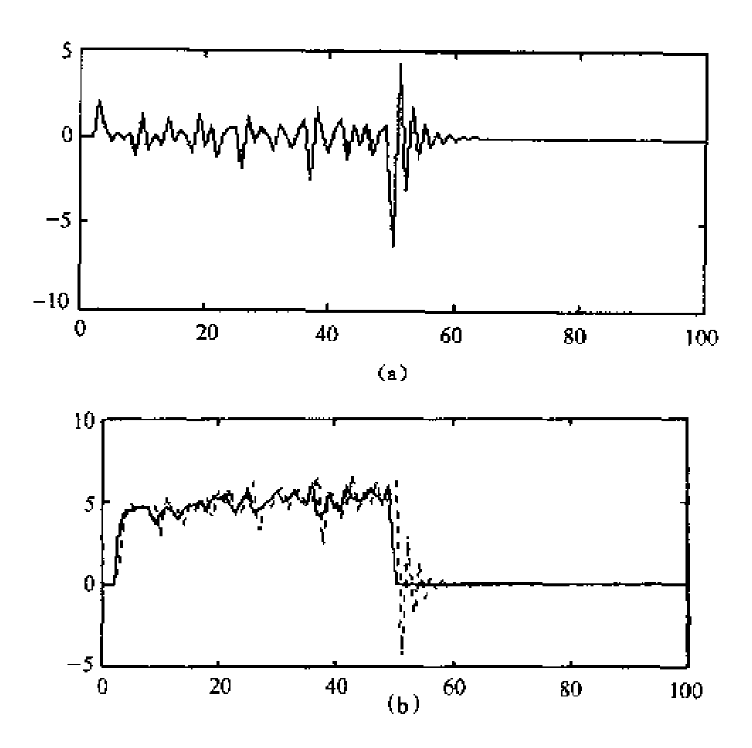


图 9.19　系统在 $k=50$ 时发生故障时的仿真结果

(a)为系统故障时的输出预报残差曲线；(b)系统故障时的实际输出(实线)和预报输出曲线(虚线)

第十章　自组织竞争人工神经网络

在生物神经系统中，存在一种“侧抑制”现象，即一个神经细胞兴奋后，通过它的分支会对周围其他神经细胞产生抑制. 这种侧抑制使神经细胞之间出现竞争，虽然开始阶段各个神经细胞都处于程度不同的兴奋状态，由于侧抑制的作用，各细胞之间相互竞争的最终结果是：兴奋作用最强的神经细胞所产生的抑制作用战胜了它周围所有其他细胞的抑制作用而“赢”了，其周围的其他神经细胞则全“输”了.

自组织竞争人工神经网络正是基于上述生物结构和现象形成的. 它是一种以无教师示教的方式进行网络训练，具有自组织功能的神经网络，网络通过自身训练，自动对输入模式进行分类. 在网络结构上，自组织竞争人工神经网络一般是由输入层和竞争层构成的两层网络，网络没有隐含层，两层之间各神经元实现双向连接，有时竞争层各神经元之间还存在横向连接. 在学习算法上，它模拟生物神经系统依靠神经元之间的兴奋、协调与抑制、竞争的作用来进行信息处理的动力学原理，指导网络的学习与工作.

自组织竞争人工神经网络的基本思想是网络竞争层各神经元竞争对输入模式的响应机会，最后仅一个神经元成为竞争的胜者，并对那些与获胜神经元有关的各连接权朝着更有利于它竞争的方向调整，这一获胜神经元就表示对输入模式的分类. 除了竞争方法外，还有通过抑制手段获胜的方法，即网络竞争层各神经元都能抑制所有其他神经元对输入模式的响应机会，从而使自己成为获胜者. 此外，还有一种侧抑制的方法，即每个神经元只抑制与自己邻近的神经元，而对远离自己的神经元则不抑制. 因此，自组织竞争人工神经网络自组织自适应的学习能力进一步拓宽了神经网络在模式识别、分类方面的应用.

10.1　两种联想学习规则

10.1.1　Instar 学习规则

由 r 个输入构成的格劳斯贝格 Instar 模型如图 10.1 所示.

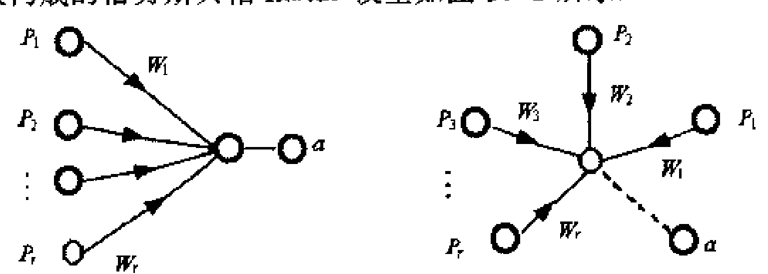


图 10.1　格劳斯贝格 Instar 模型

Instar 模型通过调节网络权值矢量 W 近似于输入矢量 P 来训练某一神经元结点只响应特定的输入矢量 P，实现其输入/输出转换的激活函数是硬限制函数. 其对权值修正

的格劳斯贝格 Instar 学习规则为

$$\Delta W(i,j) = \eta * (P(j) - W(i,j)) * a(i) \tag{10.1}$$

其中 η 为网络的学习率. 由上式可见, $\Delta W(i,j)$ 与输出成正比. 如果 Instar 模型的输出矢量 a 被某一外部方式维护高值时,那么通过不断反复地学习,权值将能够逐渐趋近于输入矢量 $P(j)$ 的值,并驱使 $\Delta W(i,j)$ 逐渐减少, 直到最终达到 $W(i,j)=P(j)$,从而使 Instar 权矢量学习了输入矢量 P, 达到了用 Instar 模型来识别一个矢量的目的. 另外, 如果 Instar 模型的输出保持为低值时, 网络权矢量被学习的可能性较小, 甚至不能被学习.

对于一个已训练过的 Instar 模型, 当输入端再次出现该学习过的输入矢量时, 其产生 1 的加权输入和; 而与学习过的矢量不相同的输入出现时, 所产生的加权输入和总是小于 1. 由此可见, Instar 加权输入和公式中的权值 W 与输入矢量 P 的点积, 反映了输入矢量与网络权矢量之间的相似度, 当相似度接近于 1 时, 表明输入矢量 P 与权矢量相似, 并通过进一步学习, 能够使权矢量对其输入矢量具有更大的相似度. 当多个相似输入矢量输入 Instar 模型时, 最终的训练结果是使网络的权矢量趋向于相似输入矢量的平均值.

10.1.2 Outstar 学习规则

由 s 个输出结点构成的格劳斯贝格 Outstar 模型如图 10.2 所示.

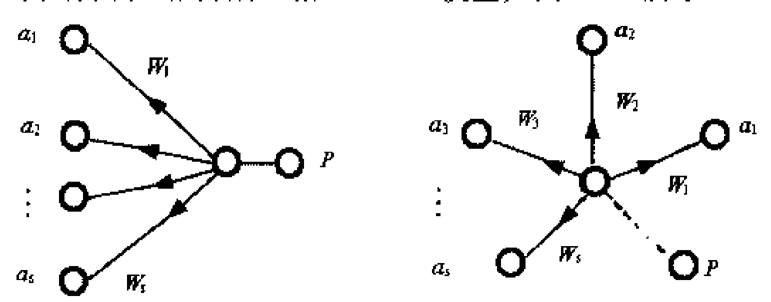


图 10.2 格劳斯贝格 Outstar 模型

Outstar 模型被训练来在一层 s 个线性神经元的输出端产生一个矢量 a, 其激活函数是线性函数,它被用来学习回忆一个矢量. Outstar 模型连接强度的变化 ΔW 是与输入矢量 P 成正比的, 其对权值修正的格劳斯贝格 Outstar 学习规则为

$$\Delta W(i,j) = \eta * (a(i) - W(i,j)) * P(j) \tag{10.2}$$

其中, η 为网络的学习率. 由上式可见, $\Delta W(i,j)$ 与输入成正比. 如果 Outstar 模型的输入矢量 $P(j)$ 被保持高值时, 那么通过不断反复地学习, 权值将能够逐渐趋近于输出矢量 $a(i)$ 的值, 并驱使 $\Delta W(i,j)$ 逐渐减少, 直到最终达到 $W(i,j)=a(i)$. 当有 r 个 Outstar 相并联,每个 Outstar 模型与 s 个线性神经元相连组成一层 Outstar 时,每当某个 Outstar 的输入结点被置为 1 时,与其相连的权值矢量就会被训练成对应的线性神经元的输出矢量. 另外, 如果 Outstar 模型的输入保持为低值时, 网络权矢量被学习的可能性较小, 甚至不能被学习与修正.

10.2　基本竞争型人工神经网络

基本竞争型人工神经网络由输入层和竞争层组成，输入层有 N 个神经元，竞争层有 M 个神经元，其网络基本结构如图 10.3 所示.

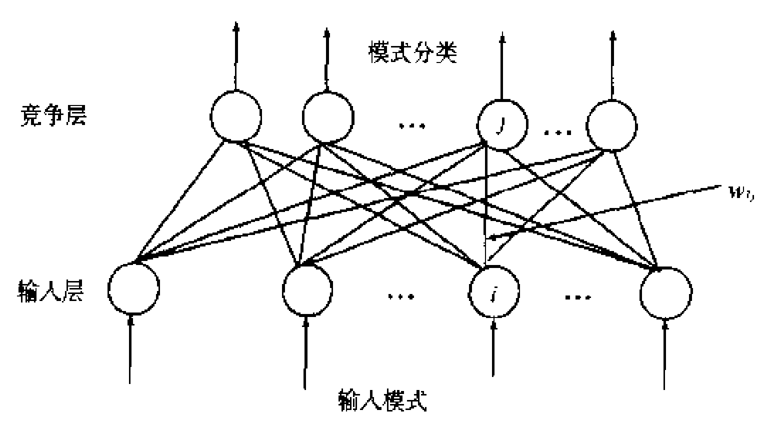


图 10.3　基本竞争网络结构

其中网络的连接权为 $\{W_{ij}\}$，$i=1, 2, \cdots, N$；$j=1, 2, \cdots, M$，且约束条件为

$$\sum_{i=1}^{N} W_{ij} = 1 \tag{10.3}$$

网络的 T 个二值输入学习模式为：$P_k=(p_1^k, p_2^k, \cdots, p_N^k)$，与其对应的竞争层输出模式为：$A_k=(a_1^k, a_2^k, \cdots, a_M^k)$，$k=1, 2, \cdots, T$.

网络的学习规则为

1）初始化. 按照式（10.3）的约束条件赋予 $\{W_{ij}\}$ 为 $[0, 1]$ 区间内的随机值，$i=1, 2, \cdots, N$；$j=1, 2, \cdots, M$.

2）任选 T 个学习模式中的一个模式 P_k 提供给网络的输入层.

3）按照下式计算竞争层各神经元的输入值 S_j

$$S_j = \sum_{i=1}^{N} W_{ij} p_i^k, \quad i = 1,2,\cdots,N \tag{10.4}$$

4）按照"胜者为王"的原则，以 S_j（$j=1, 2, \cdots, M$）中最大值所对应的神经元作为胜者，将其输出状态置为 1，而其他所有神经元的输出状态置为 0，即

$$\begin{aligned} a_j &= 1, \quad S_j > S_i (i \neq j) \\ a_i &= 0, \quad i \neq j \end{aligned} \tag{10.5}$$

如果出现 $S_j = S_i$ 的现象，则按统一约定取左边的神经元为获胜神经元.

5）与获胜神经元相连的各连接权按照下式进行修正，而其他所有连接权保持不变.

$$\begin{aligned} w_{ij} &= w_{ij} + \Delta w_{ij} \\ \Delta w_{ij} &= \eta\left(\frac{p_i^k}{m} - w_{ij}\right) \\ i &= 1,2,\cdots,N \ (0 < \eta < 1) \end{aligned} \tag{10.6}$$

其中 η 为学习系数，m 为第 k 个学习模式 $P_k=(p_1^k, p_2^k, \cdots, p_N^k)$ 中元素为 1 的个数.

6）选取另一个学习模式，返回步骤 3)，直至 T 个学习模式全部提供给网络.

7）返回步骤 2），直至各连接权的调整量变得很小为止.

以上的学习规则分析如下：

1）式（10.6）中的学习系数 η 反映了学习过程中连接权调整量的大小，η 的典型值一般为 0.01～0.03.

2）由式（10.6）可见，当 P_i 为 1 时，竞争层获胜神经元 j 与输入层神经元 i 之间的连接权 w_{ij} 在满足式（10.3）的约束条件下有 $w_{ij}<1$，所以其调整量为正，即连接权向增大的方向变化；当 p_i 为 0 时，调整量为负，即连接权向减小的方向变化. 所有的连接权始终在（0，1）之间变化.

3）当同一个学习模式反复提供给网络学习后，则这一模式前次所对应的竞争层获胜神经元的输入值 S_j 会逐渐增大，继续保持其胜者的地位. 当与这一学习模式非常接近的模式提供给网络时，也将促使同一神经元在竞争中获胜. 因此，在网络回想时，就可以根据所记忆的学习模式按照式(10.5)对输入模式作出最邻近分类，即以竞争层获胜神经元表示分类结果.

10.3　自组织特征映射神经网络

10.3.1　自组织特征映射网络的结构

自组织特征映射网络(SOM 网络)是由芬兰赫尔辛基大学神经网络专家 Kohonen 教授在 1981 年提出的，这种网络模拟大脑神经系统自组织特征映射的功能，它是一种竞争式学习网络，在学习中能无监督地进行自组织学习.

脑神经学的研究表明，人脑中大量的神经元处于空间不同区域，有着不同的功能. 它们敏感着各自的输入信息模式的不同特征，这样就形成了大脑各种不同感知路径. 在大脑皮层中，神经元的输入信号一部分来自感觉组织或其他区域的外部输入信号，另一部分来自同一区域的反馈信号. 神经元之间的信息交互具有的共同特征是，最邻近的两个神经元互相激励而兴奋，较远的相互抑制，更远的又是弱激励，这种局部作用的交互关系如图 10.4 所示.

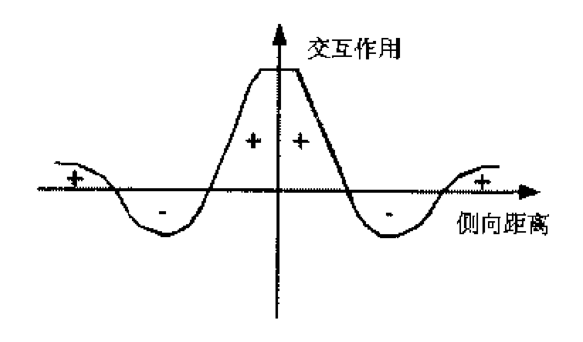


图 10.4　侧向交互作用关系

Kohonen 网络结构如图 10.5 所示，它由输入层和竞争层组成.

输入层神经元数为 n，竞争层由 $M=m^2$ 个神经元组成，且构成一个二维平面阵列. 输入层与竞争层之间实行全互连接，有时竞争层各神经元之间还实行侧抑制连接. 网络中

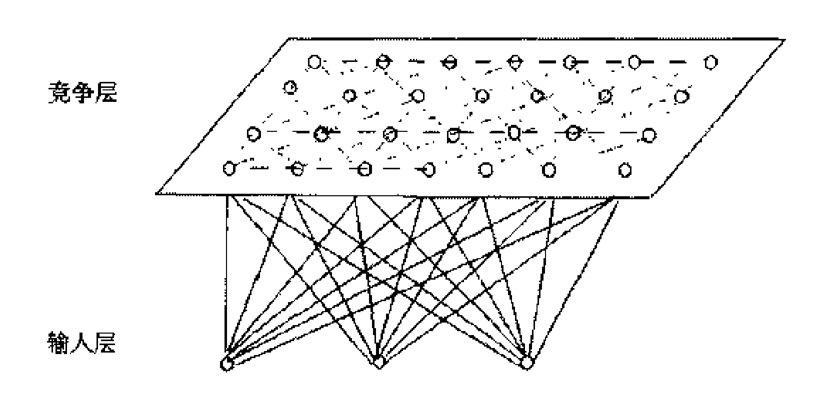


图 10.5　自组织特征映射网络结构

有两种连接权值，一种是神经元对外部输入反应的连接权值，另一种是神经元之间的连接权值，它的大小控制着神经元之间的交互作用的大小.

自组织特征映射算法是一种无教师示教的聚类方法，它能将任意维输入模式在输出层映射成一维或二维离散图形，并保持其拓扑结构不变. 即在无教师示教的情况下，通过对输入模式的自组织学习，在竞争层将分类结果表示出来. 此外，网络通过对输入模式的反复学习，可以使连接权矢量空间分布密度与输入模式的概率分布趋于一致，即连接权矢量空间分布能反映输入模式的统计特征.

Kohonen 认为，一个神经网络接受外界输入模式，将分成不同区域，各区域中邻近的神经元通过交互作用，相互竞争，自适应地形成了对输入模式的不同响应检测器. 因此，Kohonen 网络在结构上模拟了大脑皮层中神经元呈二维空间点阵的结构，在功能上通过网络中神经元间的交互作用和相互竞争，模拟了大脑信息处理的聚类功能、自组织、自学习功能.

10.3.2　自组织映射网络的学习及工作规则

将图 10.5 所示的自组织特征映射网络结构中各输入神经元与竞争层神经元 j 的连接情况抽出，如图 10.6 所示.

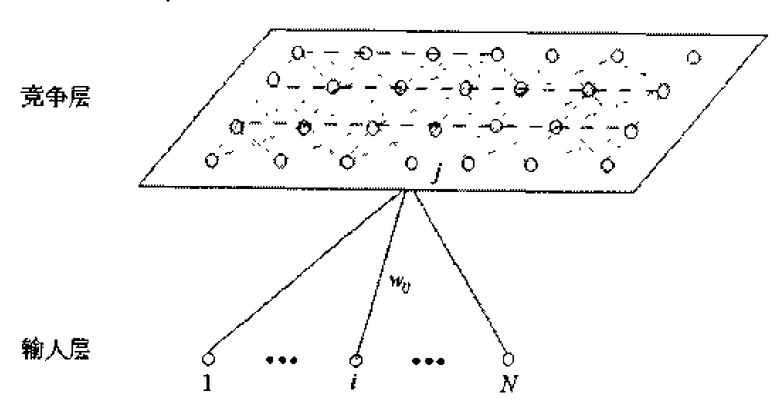


图 10.6　输入神经元与竞争层神经元 j 的连接示意

设网络的输入模式为，$P_k=(p_1^k, p_2^k, \cdots, p_N^k)$，$k=1, 2, \cdots, q$. 竞争层神经元矢量为 $A_j=(a_{j1}, a_{j2}, \cdots, a_{jm})$，$j=1, 2, \cdots, m$. 其中，$P_k$ 为连续值，A_j 为数字量. 竞争层神经元 j 与输入层神经元之间的连接权矢量为 $W_j=(w_{j1}, w_{j2}, \cdots, w_{jN})$，$i=1, 2,$

…，N；$j=1$，2，…，M.

Kohonen 学习规则是由 Instar 规则发展而来的. 对于其值为 0 或 1 的 Instar 模型输出，只对输出为 1 的 Instar 权矩阵进行修正，即学习规则只应用于输出为 1 的 Instar 上，将 Instar 模型的学习规则中的输出 a 取值 1，则可导出 Kohonen 规则. Kohonen 网络的自组织学习过程包括两个部分：一是选择最佳匹配神经元，二是权矢量自适应变化的更新过程.

算法的第一步是寻求一个评价函数以确定输入矢量 P_k 与连接权矢量 W_j 的最佳匹配，即决定竞争层中的获胜神经元 g. 评价函数可以采用内积 $W_j^T P_k$，对于所有的 j 比较各个内积，其值最大者所对应的神经元，即为获胜神经元 g，其最大处正是由于神经元的不断交互作用所形成的"气泡"中心. 同时，如果对输入矢量 P_k 与连接权矢量 W_j 进行归一化处理，则内积最大也等效于两个矢量的欧氏距离最小. 这个最小的距离确定了神经元 g 在竞争中获胜. 当网络训练好之后，如果同样的输入模式出现时，某个神经元就兴奋起来，表示该神经元已经认识了这个模式.

当某一输入与被选神经元 j 的权值 W_j 有差异时，除该权值修正外，被选神经元的邻域 N_j 中的其他神经元也将根据它们的误差以及按照距离的大小做适当的调整，越靠近 j 的神经元调整得就越多，这样形成的邻域关系使得输入模式相近时，对应的神经元在位置上也靠近，这就是 Kohonen 网络权矢量的自适应更新过程.

Kohonen 网络的自组织学习过程也可以描述为：对于每一个网络的输入，只调整一部分权值，使权向量更接近或更偏离输入矢量，这一调整过程，就是竞争学习. 随着不断学习，所有权矢量都在输入矢量空间相互分离，形成了各自代表输入空间的一类模式，这就是 Kohonen 网络的特征自动识别的聚类功能.

网络的学习及工作规则为

1）初始化. 将网络的连接权 $\{w_{ij}\}$ 赋予 [0，1] 区间内的随机值，$i=1$，2，…，N；$j=1$，2，…，M. 确定学习率 $\eta(t)$ 的初始值 $\eta(0)$ $(0<\eta(0)<1)$；确定邻域 $N_g(t)$ 的初始值 $N_g(0)$. 邻域 $N_g(t)$ 是指以步骤 4）确定的获胜神经元 g 为中心，且包含若干神经元的区域范围. 这个区域一般是均匀对称的，最典型的是正方形或圆形区域，如图 10.7 所示. $N_g(t)$ 的值表示在第 t 次学习过程中邻域中所包含的神经元的个数；确定总的学习次数 T.

2）任选 q 个学习模式中的一个模式 P_k 提供给网络的输入层，并进行归一化处理.

$$\overline{P_k} = \frac{P_k}{\| P_k \|} = \frac{(p_1^k, p_2^k, \cdots, p_n^k)}{[(p_1^k)^2 + (p_2^k)^2 + \cdots + (p_n^k)^2]^{1/2}} \tag{10.7}$$

3）对连接权矢量 $W_j=(w_{j1}, w_{j2}, \cdots, w_{jN})$ 进行归一化处理，计算 $\overline{W_j}$ 与 $\overline{P_k}$ 之间的欧氏距离

$$\overline{w_j} = \frac{w_j}{\| w_j \|} = \frac{(w_{j1}, w_{j2}, \cdots, w_{jn})}{[(w_{j1})^2 + (w_{j2})^2 + \cdots + (w_{jn})^2]^{1/2}} \tag{10.8}$$

$$d_j = \left[\sum_{i=1}^{N} (\overline{p_i^k} - \overline{w_{ji}})^2 \right]^{1/2}, j = 1, 2, \cdots, M \tag{10.9}$$

4）找出最小距离 d_g，确定获胜神经元 g

$$d_g = \min[d_j], \qquad j = 1, 2, \cdots, M \tag{10.10}$$

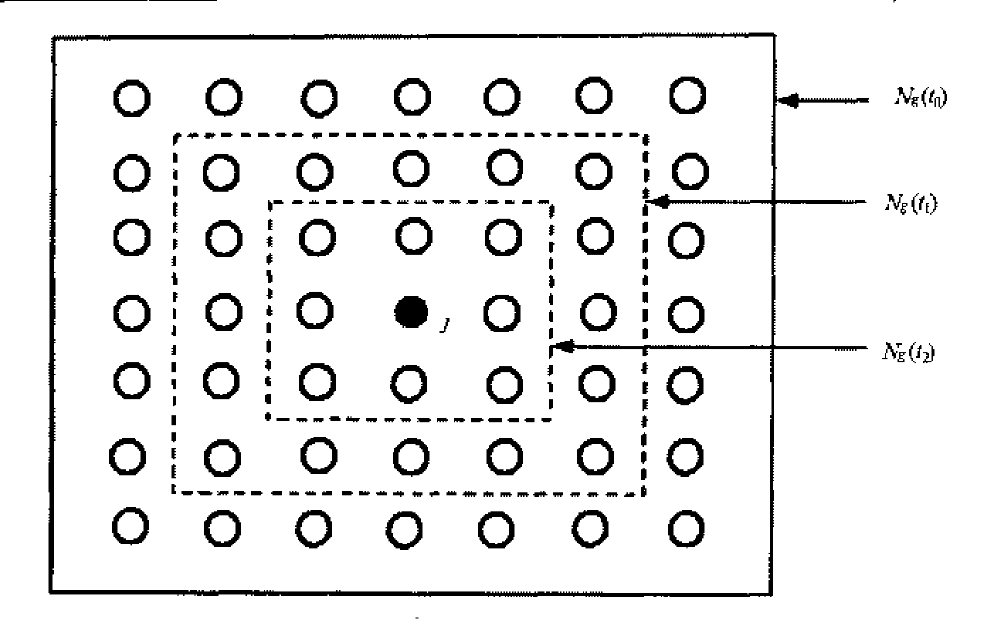


图 10.7　被选神经元 j 及其邻域变化

$(t_0<t_1<t_2)$

5）进行连接权的调整. 对竞争层邻域 $N_g(t)$ 内所有神经元与输入层神经元之间的连接权进行修正.

$$\overline{w_{jt}(t+1)} = \overline{w_{jt}(t)} + \eta(t) \cdot [\overline{p_i^k} - \overline{w_{jt}(t)}] \tag{10.11}$$

$$j \in N_g(t)\ , \qquad j = 1,2,\cdots,M \qquad (0 < \eta(t) < 1)$$

其中 η (t) 为 t 时刻的学习率.

6）选取另一个学习模式提供给网络的输入层，返回步骤 3)，直至 q 个学习模式全部提供给网络.

7）更新学习率 $\eta(t)$ 及邻域 $N_g(t)$

$$\eta(t) = \eta(0)\left(1 - \frac{t}{T}\right) \tag{10.12}$$

其中 $\eta(0)$ 为初始学习率，t 为学习次数，T 为总的学习次数.

设竞争层某神经元 g 在二维阵列中的坐标值为 (x_g, y_g)，则邻域的范围是以点 $(x_g+N_g(t), y_g+N_g(t))$ 和点 $(x_g-N_g(t), y_g-N_g(t))$ 为右上角和左下角的正方形. 其修正公式为

$$N_g(t) = \mathrm{INT}\left[N_g(0)\left(1 - \frac{t}{T}\right)\right] \tag{10.13}$$

式中 INT $[x]$ 为取整符号，$N_g(0)$ 为 $N_g(t)$ 的初始值.

8）令 $t=t+1$，返回步骤 2)，直至 $t=T$ 为止.

以上的学习规则分析如下：

1）关于学习率 η 的选择. 可以把网络学习过程分为两个阶段. 第一阶段为粗学习和粗调整阶段. 在这一阶段内，各随机方向的连接权矢量朝着输入模式的方向进行调整，并大致确定各输入模式在竞争层中所对应的映射位置. 一般此阶段的 $\eta>0.5$. 一旦各输入模式有了相对的映射位置后，则转入精学习和细调整阶段. 在这一阶段内，网络学习集中在对较小范围内的连接权进行调整，学习率应随着学习的进行不断减小. 一般此阶段

学习率的初值选为 0.5.

2）连接权矢量初始值的确定. 一般学习规则是将网络的连接权 $\{w_{ij}\}$ 赋予 [0, 1] 区间内的随机值，但在实际应用中，这种初始化方法会出现网络学习时间过长，甚至无法收敛的现象. 由于连接权矢量初始状态最理想的分布是其方向与各个输入模式的方向一致，因此在连接权初始化时，应该尽可能使其初始状态与输入模式处于一种相互容易接近的状态. 常用的方法是将所有连接权矢量赋予相同的初值，这样可以减少输入模式在最初阶段对连接权矢量的挑选余地，增加每一个连接权矢量被选中的机会，尽可能快地校正连接权矢量与输入模式之间的方向偏差.

3）邻域的作用与更新. 在自组织特征映射网络中，模拟人脑细胞受外界信息刺激产生兴奋与抑制的变化规律是通过邻域的作用来体现的. 邻域规定了与获胜神经元 g 同时进行连接权调整的神经元范围. 在学习初始阶段，$N_g(t)$包含的范围较大，一般为竞争层阵列幅度的 1/3～1/2，甚至可以覆盖整个竞争层. 随着学习的深入，$N_g(t)$的范围逐渐减小，最后达到预定的范围.

4）网络的回想. 自组织映射网络经学习后可以按照下式进行回想：

$$\begin{aligned} &a_g = 1, \text{当 } d_g = \min_{j=1}^{M}[d_j] \\ &a_i = 0, i = 1,2,\cdots,M, i \neq g \end{aligned} \tag{10.14}$$

因此，将需要分类的输入模式提供给网络的输入层，按照上述方法寻找出竞争层中连接权矢量与输入模式最接近的神经元 g，此时神经元 g 有最大的激和值 1，而其他神经元被抑制而取 0 值. 这时神经元 g 的状态即表示对输入模式的分类.

10.4 自组织竞争人工神经网络工具箱函数

MATLAB 提供了许多进行神经网络设计和分析的工具箱函数，有关这些函数的使用可通过 help 命令得到. 表 10.1 给出了与自组织竞争人工神经网络有关的工具函数. 由于本章所介绍的工具函数包含 MATLAB5.1 和 MATLAB5.3 版本，因此在备注栏加以说明.

表 10.1 自组织神经网络工具箱函数

函数名称		功能	备注
创建网络	newc	创建一个竞争层	MATLAB5.3
	newsom	创建一个自组织特征映射	MATLAB5.3
距离函数	dist	欧氏距离权值函数	MATLAB5.3
	mandist	Manhattan 距离权值函数	MATLAB5.3
	linkdist	Link 距离函数	MATLAB5.3
初始化函数	initc	初始化竞争层	MATLAB5.1
	initsm	初始化自组织特征映射网络	MATLAB5.1
	init	初始化一个神经网络	MATLAB5.3
	midpoint	中点权值初始化函数	MATLAB5.3
学习函数	learnk	Kohonen 权值学习函数	MATLAB5.3
	learnis	Instar 权值学习函数	MATLAB5.1
	learnos	Outstar 权值学习函数	MATLAB5.1
	learnsom	自组织特征映射权值学习函数	MATLAB5.3

续表

函数名称		功能	备注
训练函数	trainc	训练竞争层	MATLAB5.1
	trainsm	利用 Kohonen 规则训练自组织特征映射	MATLAB5.1
	train	训练一个神经网络	MATLAB5.3
仿真函数	simuc	竞争层仿真	MATLAB5.1
	simusm	自组织特征映射网络仿真	MATLAB5.1
	sim	仿真一个神经网络	MATLAB5.3
邻域函数	nbdist	用矢量距离表示的邻域矩阵	MATLAB5.1
	nbgrid	用栅格距离表示的邻域矩阵	MATLAB5.1
	nbman	用 Manhattan 距离表示的邻域矩阵	MATLAB5.1
权值函数	negdist	对输入矢量进行加权计算	MATLAB5.3
网络输入函数	netsum	计算网络输入矢量和	MATLAB5.3
传递函数	compet	竞争传递函数	MATLAB5.3
绘图函数	plotsm	绘制自组织特征映射网络的权值矢量	MATLAB5.1
	plotsom	绘制自组织特征映射网络	MATLAB5.3

1. 创建网络

(1) newc () 用于创建一个竞争层.

语法格式为 net=newc (P, S, KLR)

newc () 函数返回一个新的竞争层. 其中, 输入矢量矩阵P为R×2维矩阵, R为输入矢量的个数, 并且在矩阵P中必须指明每一个输入矢量的最大、最小值范围, 例如, 网络有两个输入矢量, 其变化范围分别为 [0, 1] 和 [−1, 1], 则P= [0 1; −1 1]. 同时, 变量S表示神经元的个数; 变量KLR表示Kohonen学习率, 其缺省值为0.01.

例10.1 对一个具有多输入模式的输入矢量, 可以创建一个竞争层, 在网络训练中, 竞争层中的神经元相互竞争, 以适应当前的输入样本. 最终胜利的神经元就可以代表当前的输入样本的分类模式. 因此, 竞争层可以完成对输入模式的分类.

```
P=[0.1 0.8 0.1 0.9;0.2 0.9 0.1 0.8]; %四输入模式的输入矢量
net = newc([0 1; 0 1],2);       %创建一个竞争层,对输入模式分为两类
net = train(net,P);             %网络训练
Y = sim(net,P)                  %对输入矢量仿真
Yc = vec2ind(Y)                 %对输出矢量转换为分类模式
```

运行上述程序, 其结果为

```
Y =
    (2, 1)      1
    (1, 2)      1
    (2, 3)      1
    (1, 4)      1
Yc =
    2    1    2    1
```

(2) newsom () 创建一个自组织特征映射.

语法格式为 newsom (P, [D1, D2, ..., Di]) 返回一个新的自组织特征映射.

newsom () 函数返回一个新的自组织特征映射. 其中, 输入矢量矩阵 P 为 R×2 维矩阵, R 为输入矢量的个数. 并且在矩阵 P 中必须指明每一个输入矢量的最大、最小值范围; i 表示网络层的维数大小.

例 10.2 创建一个自组织特征映射, 输入矢量分布在一个二维输入空间, 其变化范围分别为 [0 2] 和 [0 1], 自组织特征映射网络维数为 [3 5], 用 plotsom () 函数绘制自组织特征映射.

```
P = [rand(1,400) * 2; rand(1,400)];
net = newsom([0 2; 0 1],[3 5]);
plotsom(net.layers{1}.positions)
```

运行结果如图 10.8 所示, 从图中可以看出, 圆点表示神经元的位置, 它们通过欧氏距离 1 相连.

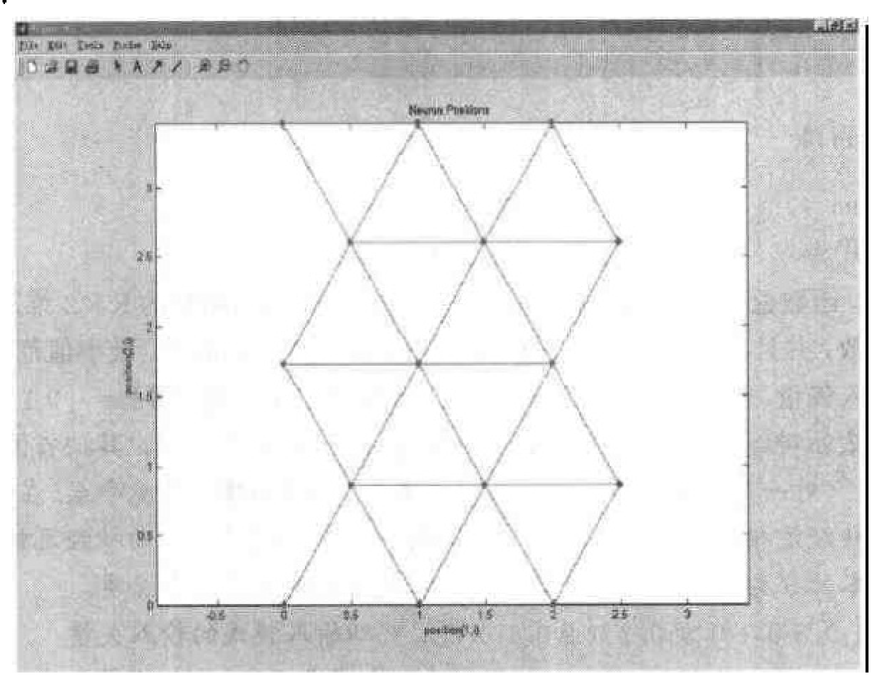


图 10.8 范例 10.2

2. 距离函数

(1) dist () 欧氏距离权值函数

语法格式为 Z = dist(W,P) 或 D = dist(pos)

dist (W, P) 函数是一个欧氏距离权值函数, 它对输入进行加权, 得到被加权的输入. 其中, W 是权值函数, P 是输入矢量矩阵, 函数返回一个矢量距离矩阵. 一般而言, 两个矢量 X 和 Y 之间的欧氏距离 d 定义为

d = sum((x - y).^2).^0.5

dist (pos) 函数也可以作为一个阶层距离函数, 用于查找某一层神经网络中的所有神经元之间的欧氏距离, 函数也返回一个距离矩阵.

例 10.3 任意给定输入矢量和权值矩阵, 可利用 dist () 函数计算其欧氏距离.

```
W = rand (4, 3);
P = rand (3, 1);
Z = dist (W, P)
```

程序运行结果为

```
Z =
    0.3962
    0.7123
    0.7050
    0.8759
```

(2) mandist ()　　Manhattan 距离权值函数.

语法格式为　Z = mandist(W,P) 或 D = mandist(pos);

mandist（W，P）函数是一个 Manhattan 距离权值函数，它对输入进行加权，得到被加权的输入. 其中，W 是权值函数，P 是输入矢量矩阵，函数返回一个矢量距离矩阵. 一般而言，两个矢量 X 和 Y 之间的 Manhattan 距离 d 定义为

d = sum(abs(x − y))

dist（pos）函数也可以作为一个阶层距离函数，用于查找某一层神经网络中的所有神经元之间的 Manhattan 距离，函数也返回一个距离矩阵.

例 10.4　任意给定输入矢量和权值矩阵，可利用 mandist () 函数计算其 Manhattan 距离.

```
W = rand (4, 3);
P = rand (3, 1);
Z = mandist (W, P)
```

程序运行结果为

```
Z =
    0.8561
    1.5249
    0.6963
    1.6585
```

(3) linkdist ()　　Link 距离函数

语法格式为 d = linkdist (pos)

linkdist (pos) 函数是一个阶层距离函数，用于查找某一层神经网络中的所有神经元之间的距离，函数返回一个距离矩阵. 一般而言，对于 S 个矢量中，两个位置矢量 Pi 和 Pj 之间的 Link 距离定义为

$$
Dij\begin{cases} = 0,\ i=j \\ = 1,\ (\text{sum}((Pi-Pj).\hat{}2)).\hat{}0.5)\leqslant 1 \\ = 2,\ k \text{ 存在，且 } Dik=Dkj=1 \\ = 3,\ k1,\ k2 \text{ 存在，且 } Dik1 = Dk1k2 = Dk2j = 1 \\ = N,\ k1\ldots kn \text{ 存在，且 } Dik1 = Dk1k2 = \ldots = Dknj = 1 \\ = S,\ \text{其他} \end{cases}
$$

例 10.5 为分布在三维空间里的 10 个神经元任意定义一个位置矩阵，可以用 linkdist () 函数查找其 Link 距离.

```
pos = rand (3, 10);
D = linkdist (pos)
```

程序运行结果为

```
D =
    0   1   1   1   2   2   1   1   1   1
    1   0   1   1   1   1   1   1   1   1
    1   1   0   1   2   1   1   1   2   2
    1   1   1   0   1   1   1   1   1   1
    2   1   2   1   0   1   2   1   1   1
    2   1   1   1   1   0   1   1   1   1
    1   1   1   1   2   1   0   1   1   1
    1   1   1   1   1   1   1   0   1   1
    1   1   2   1   1   1   1   1   0   1
    1   1   2   1   1   1   1   1   1   0
```

3. 初始化函数

(1) initc () 初始化竞争层

语法格式为 w=initc (p, s)

initc () 函数返回竞争层的权值. 其中，p 为输入样本矢量矩阵，s 为神经元数. 必须注意，输入矢量 p 的第 i 行需包含有网络输入 i 的最小、最大值，这样才能正确初始化权值.

例 10.6 利用 initc () 函数初始化竞争层网络

```
p= [-3 3; 0 6];
w=initc (p, 3)
```

其程序运行结果如下：

```
w =
    0   3
    0   3
    0   3
```

(2) initsm () 初始化自组织特征映射网络

语法格式为 w=initsm (p, s)

initsm () 函数返回自组织特征映射网络的权值，其中，p 为输入矢量矩阵，s 为神经元数目. 必须注意，输入矢量 p 的第 i 行需包含有网络输入 i 的最小、最大值，这样才能正确初始化权值.

例 10.7 利用 initsm () 函数初始化自组织特征映射网络。

```
p= [-4 4; 0 9];
w=initsm (p, 3)
```

其程序运行结果如下：

```
w =
     0    4.5000
     0    4.5000
     0    4.5000
```

(3) init ()　初始化一个神经网络

语法格式为　net = init (net)

init () 函数返回一个根据网络初始化函数更新了权值和偏差的神经网络.

例 10.8　对一个利用 newc () 函数创建的竞争层网络，可用 init () 函数进行初始化.

```
net=newc([0 1;0 1],8,0.1);
net=init(net);
net.iw{1,1}
net.b{1}
```

程序运行结果显示了初始化后的网络权值和偏差.

```
ans =
     0.5000    0.5000
     0.5000    0.5000
     0.5000    0.5000
     0.5000    0.5000
     0.5000    0.5000
     0.5000    0.5000
     0.5000    0.5000
     0.5000    0.5000
ans =
     21.7463
     21.7463
     21.7463
     21.7463
     21.7463
     21.7463
     21.7463
     21.7463
```

(4) midpoint()　中点权值初始化函数

语法格式为　w=midpoint(S,P)

在 midpoint (S, P) 函数中，S 为神经元的个数，P 为 R×2 维矩阵，它包含每个输入中的最小值和最大值，函数返回一个 S×R 维权值矩阵. 它每个值设定为(Pmin+Pmax)/2. 因为网络输入很可能出现在中间区，因此，如果竞争层和自组织网络的初始权值选择在输入空间的中间区，则其学习会更加有效.

例 10.9　利用 midpoint()函数初始化权值.

```
P= [0 1; -2 2];
W=midpoint(3,P)
```

程序运行结果如下：

```
W =
     0.5000      0
     0.5000      0
     0.5000      0
```

4. 学习函数

(1) learnk () Kohonen 权值学习函数

语法格式为 dW = learnk (W, P, A, lr)

learnk () 函数在给定输入矢量矩阵 P、输出矢量矩阵 A、学习率 lr 和当前权值矩阵 W 的情况下，返回网络层的权变化矩阵. Learnk () 函数依据 kohonen 相关准则计算网络层的权变化矩阵. 其学习通过调整神经元的权值等于当前输入矢量，使神经元存贮输入矢量，用于以后的识别. 即

$$\Delta W(i,j) = lr * (P(j) - \Delta W(i,j))$$

例 10.10 一个具有两个输入矢量、三个神经元的自组织网络，在给定其随机输入矩阵 P，输出矩阵 A，权值矩阵 W 和学习率后，可以用 learnk () 函数计算其网络层的权变化矩阵.

```
p = rand (2, 1);
a = rand (3, 1);
w = rand (3, 2);
lr = 0.5;
dw = learnk (w, p, a, lr)
```

计算结果如下：

```
dw =
      0.0940    -0.2951
      0.2468    -0.1068
      0.4658    -0.1921
```

(2) learnis () Instar 权值学习函数

语法格式为 dW = learnis (W, P, A, lr)

learnis () 函数在给定输入矢量矩阵 P、输出矢量矩阵 A、学习率 lr 和当前权值矩阵 W 的情况下，返回网络层的权变化矩阵. Learnis () 函数依据 Instar 相关准则计算网络层的权变化矩阵. 其学习用一个正比于神经元输出的学习率来调整权值，学习一个新的矢量使之等于当前输入. 这样，任何使 Instar 层引起高输出的变化，都会导致网络根据当前的输入矢量学习这种变化. 最终，相同的输入使网络有明显不同的输出. 即

$$\Delta W(ij) = lr * (P(j) - W(i,j)) * a(i)$$

例 10.11 一个具有两个输入矢量、三个神经元的自组织网络，在给定其随机输入矩阵 P，输出矩阵 A，权值矩阵 W 和学习率后，可以用 learnis () 函数计算其网络层的权

变化矩阵.

```
P = rand (2, 1);
a = rand (3, 1);
w = rand (3, 2);
lr = 0.5;
dw = learnis (w, p, a, lr)
```

计算结果如下:

```
dw =
      0.854      0.0363
      0.854      0.1189
      0.854      0.1376
```

(3) learnos () Outstar 权值学习函数

语法格式为 dW = learnos (W, P, A, lr)

learnos () 函数在给定输入矢量矩阵P、输出矢量矩阵A、学习率lr 和当前权值矩阵W 的情况下，返回网络层的权修改矩阵. Learnos () 函数依据Outstar 相关准则计算网络层的权修改矩阵. Outstar 网络层的权可以看作是与网络层的输入矢量一样多的长期存贮器. 通常，Outstar 层是线性的，允许输入权值按线性层学习输入矢量. 因此，存贮在输入权值中的矢量可通过激和该输入而得到. 即

$$\Delta W(i,j) = lr * (a(i) - W(i,j)) * P(j)$$

例10.12 一个自组织网络，在给定其随机输入矩阵P，输出矩阵A，权值矩阵W 和学习率后，可以用learnos () 函数计算其网络层的权变化矩阵.

```
P = rand (3, 2);
a = rand (3, 2);
w = rand (3, 3);
lr = 0.5;
dw = learnos (w, p, a, lr)
```

计算结果如下:

```
dw =
    -0.0990    -0.0673    -0.3756
    -0.2352     0.0938    -0.6251
     0.0159    -0.0981     0.0448
```

(4) learnsom () 自组织特征映射权值学习函数

语法格式为 [dW, LS] = learnsom (W, P, A, D, LP, LS)

Learnsom () 函数中，W 为权值矩阵或者为偏差矢量. P 为输入矢量，A 为输出矢量，D 为神经元距离，LP 为学习参数，LS 为学习状态，初始化为空矩阵. 网络学习是根据learnsom () 函数所给出的学习参数开始的，其正常状态学习率LP.order_lr 缺省值为0.9，正常状态学习步数LP.order_steps 缺省值为1000，调整状态学习率LP.tune_lr 缺省值为0.02，调整状态邻域距离LP.tune_nd 缺省值为1. 返回dW 为权值或者偏差变化矩阵，LS 为新的学习状态.

Learnsom () 函数根据神经元的输入矩阵 P、激励矩阵 A2 和学习率 lr 计算神经元的权变化矩阵. 即：$\Delta W(i,j)=lr * A2 * (P(j)-W(i,j))$. 其中，激励矩阵 A2 与网络层的输出矩阵 A、神经元距离 D 和当前的邻域尺寸 nd 的关系为

$$a2(i,q)\begin{cases}=1, & \text{if } a(i,q)=1\\ =0.5, & \text{if } a(j,q)=1 \text{ and } D(i,j)<=nd\\ =0, & \text{otherwise}\end{cases}$$

在网络处于正常状态和调整状态时，学习率 lr 和邻域尺寸 nd 都得到更新. 正常状态的持续时间由 LP.order_steps 定义，在这期间，学习率 LR 从 LP.order_lr 调整到 LP.tune_lr，邻域尺寸从最大神经元距离下调到 1. 神经元的权值按照期望的方向在适应神经元位置的输入空间建立次序. 在调整状态，学习率 lr 慢慢从 LP.tune_lr 减少，邻域距离 nd 常常被设为 LP.tune_nd. 在这期间，神经元的权值按照期望的方向从输入空间伸展，直到保留到他们在正常状态时所建立的拓扑次序.

例 10.13 一个具有两个输入矢量、6 个神经元的自组织网络，在给定其随机输入矩阵 P，输出矩阵 A，权值矩阵 W 和学习率后，可以用 learnsom() 函数计算其权值变化矩阵.

```
P = rand(2,1);
a = rand(6,1);
w = rand(6,2);
lp.order_lr = 0.9;
lp.order_steps = 1000;
lp.tune_lr = 0.02;
lp.tune_nd = 1;
ls = [];
[dW,ls] = learnsom(w,p,a,d,lp,ls)
```

计算结果如下：

```
dW =
      0.6472    -0.1123
     -0.1585    -0.4588
      0.6472     0.0278
      0.6472     0.2678
      0.6472     0.4501
     -0.1375     0.4069
```

5. 训练函数

(1) trainsm () 利用 Kohonen 规则训练自组织特征映射网络

语法格式为 w=trainsm(w,b,p,tp)

trainsm () 函数在训练参数 tp 的控制下，对输入矢量为 p 时的初始权值进行训练，返回一个新的权值矩阵 w. 其中，训练参数 tp 可设定为

tp(1) 表示两次更新显示的迭代次数，缺省值为 25

tp(2) 表示训练迭代的最大次数，缺省值为 100

tp(3) 表示初始学习率，缺省值为 1

自组织特征映射网络由一层分布在一维或多维空间中的神经元构成，在任何时候，只有网络输入最大的神经元输出为 1，相邻的神经元输出为 0.5，其余的神经元输出为 0. 在网络训练时，从 p 中随机地选取输入矢量加到网络中，直到满足最大迭代次数. 训练结束后，在自组织特征映射网络中，神经元把自己组织起来，以便使每个神经元响应不同的输入矢量. 另外，相邻的神经元响应相似的输入矢量. 输入矢量出现较频繁的区域由较多的神经元来分类，而输入相对少的区域则所需的神经元较少. 由此，自组织特征映射网络可以根据输入的拓扑结构和概率分布进行分类.

例 10.14 任意创建一个具有 100 个元素的输入矢量，构造一个排列在 3×3 栅格上由 9 个神经元组成的自组织特征映射网络，对网络训练 400 次，观察网络的自组织能力.

```
p=rand(2,100);
w=initsm(p,9);
m=nbman(3,3);
w=trainsm(w,m,p,[20 400])
```

网络训练结束，返回新的权值矩阵如图 10.9 所示.

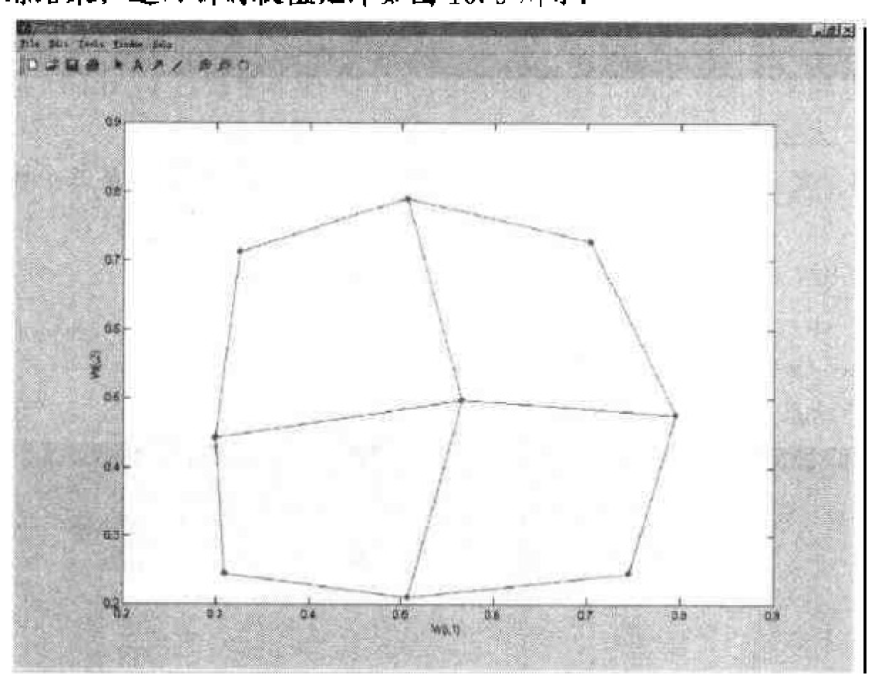


图 10.9 范例 10.14

```
w =

    0.3097    0.2442
    0.2992    0.4425
    0.3257    0.7125
    0.5060    0.2096
    0.5651    0.4973
    0.5050    0.7910
    0.7442    0.2444
```

0.7947   0.4763

0.7031   0.7282

(2) trainc()　训练竞争层，以便对一组输入矢量进行分类.

语法格式为　trainc(w,p,tp)

trainc () 函数在训练参数 tp 的控制下，通过调整初始权值矩阵 w 和偏差矢量，返回相应于随机顺序选取的输入矢量 p 的新的权值矩阵. 其中，训练参数 tp 可设定为

tp(1) 表示两次更新显示的迭代次数，缺省值为 25

tp(2) 表示训练迭代的最大次数，缺省值为 100

tp(3) 表示学习率，缺省值为 0.01

tp(4) 表示跟踪性能（从 0 到 1），缺省值为 0.999

tp(5) 表示加权性能（从 0 到 1），缺省值为 0.1

在竞争层的网络训练中，从一组输入矢量中随机选取一个矢量加到竞争网络，直到满足指定的训练次数，然后找出输入最大的神经元，并利用 Kohonen 规则更新权值. 在整个训练过程中，不断地调整偏差矢量，使得小概率输出为 1 的神经元具有较大的偏差，这可以促使那些"失败"的神经元有更多的调整机会. 同时，训练参数 tp(4) 越接近于 1，计算神经元输出均值就越慢；训练参数 tp(5) 越小，高或低的均值对神经元偏差和获胜率的影响也越小. 因此，对于稳定的学习，tp(4) 应非常接近 1，tp(5) 应非常小，然而如果 tp(4) 太接近 1，tp(5) 太小，网络的训练时间也会增加.

例 10.15　定义一个三维、标准化的二元输入矢量，训练一个单层三个神经元的竞争网络.

```
p= [0.5827 0.6496 -0.7798; 0.8127 0.7603 0.6260];
w=initc (p, 3);
w=trainc (w, p, [10 100 0.6])
```

程序运行结果如图 10.10 所示.

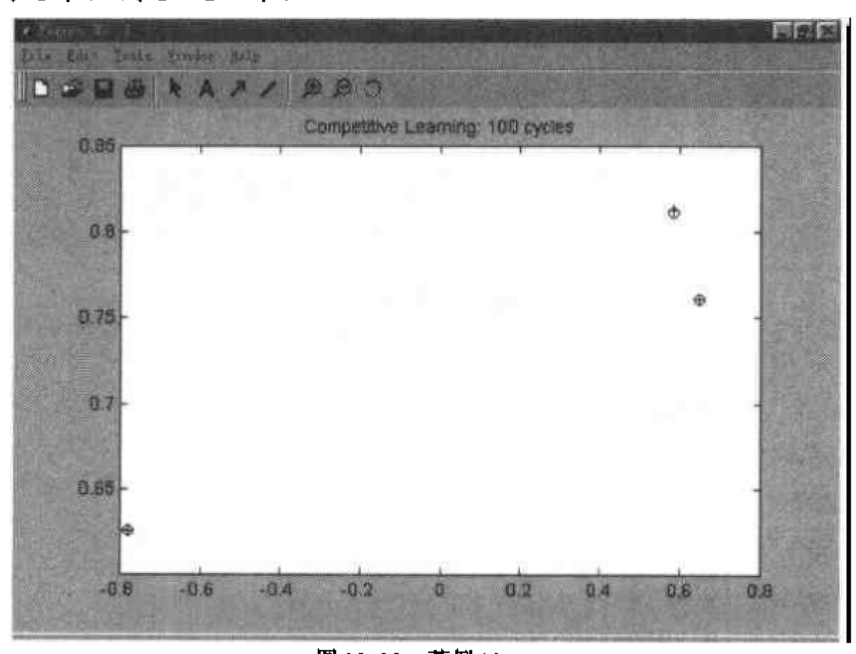


图 10.10　范例 10.15

```
w =
    -0.7798    0.6260
     0.6496    0.7603
     0.5827    0.8127
```

从上图可以看出，神经元1的权矢量学会了第二个输入矢量，神经元2的权矢量学会了第三个输入矢量，神经元3的权矢量学会了第一个输入矢量．因此，与输入矢量相似的新矢量的出现将导致相应的神经元输出为1.

(3) train()　　训练一个神经网络

语法格式为　　[net，tr] = train (NET，P，T)

train() 函数返回值 net 是训练好的神经网络，tr 是有关训练的进程记录．其参数 P 是网络输入矢量，T 是网络目标矢量.

6. 仿真函数

(1) simuc ()　　竞争层仿真

语法格式为　　simuc (p，w)

simuc(p，w) 函数返回网络层的输出，其中 p 为输入矢量矩阵，w 为竞争层的权值矩阵．一个竞争层包含一层神经元，在任何给定时间，只有网络输入最大的神经元其输出为1，其他的神经元输出为0．由于竞争层的任何一个输出向量中仅含惟一的非零值，因此 simuc() 函数的返回值是一个稀疏矩阵，这就为计算机存贮提供了方便，可以大大降低对机器内存的需求.

例 10.16　*利用 simuc 函数可以计算出竞争层对输入的响应.*

```
w=initc ( [0 2; -5 5], 4);
a=simuc ( [2; 4], w)
```

程序运行结果如下，表示神经元1有一个输出1.

```
a =
    (1, 1)        1
```

(2) simusm ()　　自组织特征映射网络仿真

语法格式为　　simusm(p,w,m)

　　　　　　　　simusm(p,w,m,n)

simusm() 函数返回自组织特征映射网络的输出．其中 p 为输入矢量矩阵、w 为权值矩阵，m 为网络的邻阵，参数 n 为邻域的大小，缺省值为1．自组织特征映射网络由分布在一维或多维空间中的神经元组成，在任何时候，只有网络输入最大的神经元有输出1，与获胜神经元相邻的神经元输出为0.5，其余神经元输出为0．因此，对于网络输入最大的神经元 i，其对应的输出为1，与之距离在1之内 ($m(i, j) \leqslant 1$) 的神经元 j，其输出为0.5.

例 10.17　*利用 simusm 函数可以计算出自组织特征映射网络对输入的响应.*

```
w=initsm([-3 3;0 6],6);
m=nbman(2,3);
a=simusm([2;5],w,m)
```

程序运行结果如下，表示神经元 1 有一个输出 1，神经元 2 和神经元 4 分别有输出 0.5.

```
a =
     (1, 1)        1.0000
     (2, 1)        0.5000
     (4, 1)        0.5000
```

(3) sim () 　**网络层仿真**

语法格式为　Y = sim(net, P)

sim(net, P) 函数对一个具有输入矢量 P 的神经网络 net 进行仿真，函数返回网络输出.

例 10.18　*对 newff() 函数定义的一个二层前向网络，可以用函数 train() 进行训练，用 sim() 函数进行仿真.*

```
p = [0 1 2 3 4 5 6 7 8];
t = [0 0.84 0.91 0.14 -0.77 -0.96 -0.28 0.66 0.99];
net = newff ( [0 8], [10 1], {'tansig' 'purelin'},'trainlm');
y1 = sim (net, p)
net.trainParam.epochs = 50;
net.trainParam.goal = 0.01;
net = train (net, p, t);
y2 = sim (net, p)
plot (p, t,'o', p, y1,'x', p, y2,'*')
```

程序运行结果如图 10.11 所示. 从图中可见，代表网络训练前的仿真输出 y1 的符号 "x" 与代表网络目标矢量的符号 "o" 不能重合，而代表网络训练后的仿真输出 y2 的符号 "*" 与代表网络目标矢量的符号 "o" 完全重合. 由此可见，网络训练性能良好，仿真输出表明了网络训练的有效性.

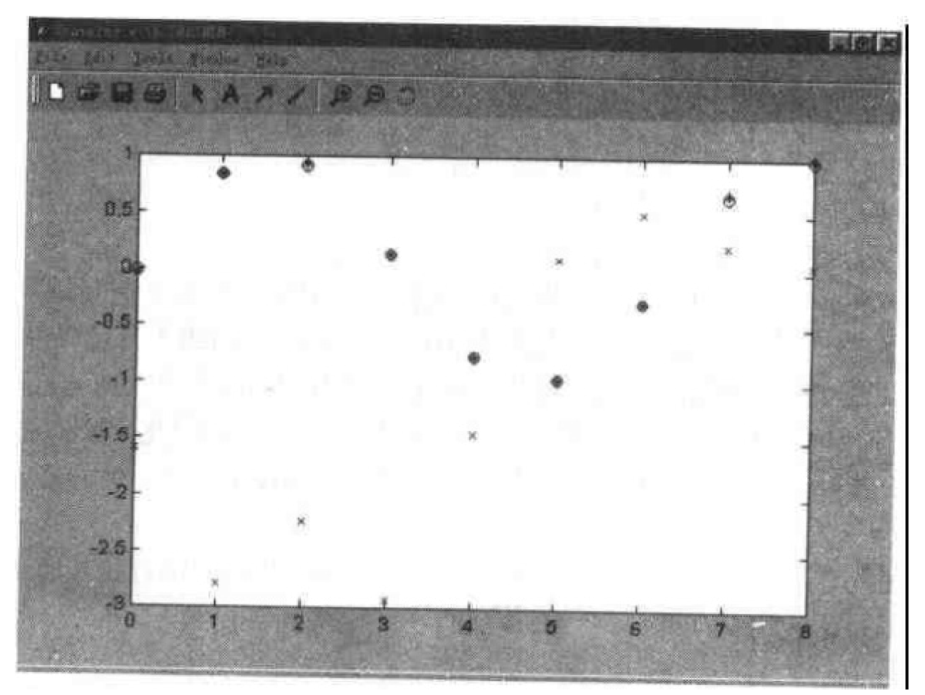


图 10.11　范例 10.18

7. 邻域函数

(1) nbdist()　　用矢量距离表示的邻域矩阵

语法格式为　nbdist (d1)

nbdist (d1, d2)

nbdist (d1, d2, …, d5)

自组织网络中的神经元可以按任何方式排列，这种排列可以用表示同一层神经元间距离的邻域阵来描述. 因此，nbdist (d1) 返回一维排列的 d1×d1 的邻域阵，表示一维中含有d1 个神经元，其元素 (i, j) 表示神经元i 与神经元j 之间的矢量距离. nbdist(d1, d2)返回二维排列的(d1 * d2)×(d1 * d2)邻阵，表示二维中包含有 d1 * d2 个神经元，其元素(i,j)表示神经元i 与神经元j 之间的矢量距离. nbdist()函数可用于每层最多有5维，最多有5层的网络，一般而言维数越少，收敛越快.

例 10.19　m=nbdist (2, 3)

程序运行结果为

m =

0	1.0000	2.0000	1.0000	1.4142	2.2361
1.0000	0	1.0000	1.4142	1.0000	1.4142
2.0000	1.0000	0	2.2361	1.4142	1.0000
1.0000	1.4142	2.2361	0	1.0000	2.0000
1.4142	1.0000	1.4142	1.0000	0	1.0000
2.2361	1.4142	1.0000	2.0000	1.0000	0

(2) nbgrid ()　　用栅格距离表示的邻域矩阵

语法格式为　nbgrid(d1)

nbgrid(d1,d2)

nbgrid(d1,d2,…,d5)

自组织网络中的神经元可以按任何方式排列，这种排列可以用表示同一层神经元间距离的邻域阵来描述，而两神经元的栅格距离是指在神经元坐标相减后的矢量中，其元素幅值的最大值. 因此，nbgrid (d1) 返回一维排列的 d1×d1 的邻域阵，表示一维中含有d1 个神经元，其元素 (i, j) 表示神经元i 与神经元j 之间的栅格距离. nbgrid (d1, d2)返回二维排列的(d1 * d2)×(d1 * d2)邻阵，表示二维中包含有 d1 * d2 个神经元，其元素(i,j)表示神经元i 与神经元j 之间的栅格距离. nbgrid()函数可用于每层最多有5维，最多有5层的网络，一般而言维数越少，收敛越快.

例 10.20　m= nbgrid(2, 3)

程序运行结果为

m =

0	1	2	1	1	2
1	0	1	1	1	1
2	1	0	2	1	1
1	1	2	0	1	2

```
1   1   1   1   0   1
2   1   1   2   1   0
```

(3) nbman() 用 Manhattan 距离表示的邻域矩阵

语法格式为 nbman(d1)

nbman(d1,d2)

nbman(d1,d2,…,d5)

自组织网络中的神经元可以按任何方式排列，这种排列可以用表示同一层神经元间距离的邻域阵来描述，而两神经元的 Manhattan 距离是指在神经元坐标相减后的矢量中，其元素绝对值之和. 因此，nbman (d1) 返回一维排列的 d1×d1 的邻域阵，表示一维中含有 d1 个神经元，其元素(i,j)表示神经元 i 与神经元 j 之间的 Manhattan 距离. Nbman (d1,d2)返回二维排列的(d1 * d2)×(d1 * d2)邻阵，表示二维中包含有 d1 * d2 个神经元，其元素(i,j)表示神经元 i 与神经元 j 之间的 Manhattan 距离. nbman()函数可用于每层最多有 5 维，最多有 5 层的网络，一般而言维数越少，收敛越快.

例 10.21 m= nbman (2, 3)

程序运行结果为

```
m =0   1   2   1   2   3
   1   0   1   2   1   2
   2   1   0   3   2   1
   1   2   3   0   1   2
   2   1   2   1   0   1
   3   2   1   2   1   0
```

8. 权值函数

negdist() 对输入矢量进行加权计算

语法格式为 Z = negdist (W, P)

negdist(W, P) 函数中，W 表示 S×R 维权值矩阵，P 表示 R×Q 维输入矢量，函数返回 S×R 维负矢量距离矩阵. 即

Z=−sqrt (sum (w−p)^2)

例 10.22 给定权值矩阵和输入矢量，可以利用 negdist (W, P) 函数计算相应的加权输入.

```
W = rand (4, 3);
P = rand (3, 1);
Z = negdist (W, P)
```

计算结果如下

```
Z =
    -1.0589
    -0.4162
    -0.7200
    -0.4760
```

9. 网络输入函数

netsum() 计算网络输入矢量和

语法格式为 N = netsum (Z1, Z2, ...)

netsum() 函数合并网络加权输入和偏差，计算输入矢量和，并返回矢量 Zi 相应的元素和.

例 10.23 给定网络加权输入矢量和偏差，可以利用 netsum() 函数计算网络输入矢量和.

```
z1 = [1 2 4; 3 4 1];
z2 = [-1 2 2; -5 -6 1];
n = netsum (z1, z2)
b = [0; -1];
m = netsum (z1, z2, concur (b, 3))
```

程序运行结果如下

```
n =
     0     4     6
    -2    -2     2
m =
     0     4     6
    -3    -3     1
```

10. 传递函数

compet()　竞争传递函数

语法格式为　a = compet (n)

　　　　　　a = compet (z, b)

compet() 函数将神经元的网络输入进行转换，使网络输入最大的神经元输出为 1，而其余的神经元输出为 0. compet(n) 函数返回一个输出矢量矩阵，其每一列矢量中仅包含一个 1，位于网络输入矢量 n 为最大的位置，而其余的元素为 0. compet(z, b) 函数用于批处理矢量、且偏差存在的情况下，偏差矢量 b 附加到加权输入矩阵 z 的每一个矢量上，形成网络输入矢量矩阵 n，然后利用竞争传递函数将输入矢量转换为输出矢量.

例 10.24 利用传递函数计算网络层的输出矢量.

```
n= [0.1; 0.4; 0.9; 0.0];
a=compet (n)
full (a)
```

从如下程序运行结果中，可以看出输出矢量在网络输入矢量的最大元素处有一个 1.

```
a =
   (3, 1)        1
ans =
     0
```

0

1

0

11. 绘图函数

(1) plotsm() 绘制自组织特征映射网络的权值矢量

语法格式为 plotsm(W, M)

plotsm(W, M) 函数给出了自组织网络的图形表示. 在 W 中的每个神经元的权值行矢量相应的坐标处绘制出一点,然后表示相邻神经元权值的点之间根据邻阵 M 用线连接起来. 即当 M(i, j) ≤1 时, 则将神经元 i 和 j 用线连接起来.

例 10.25 利用 rands 函数可给两输入的 12 个神经元产生随机权值, 并将其排列成 3×4 的栅格, 然后可绘制出这一层的权值连接图.

```
w=rands (12, 2)
m=nbman (3, 4)
plotsm (w, m)
```

绘制的权值连接图如图 10.12 所示.

```
w =
   -0.2607    0.8158
    0.3969   -0.1975
    0.7928    0.1544
   -0.7750   -0.0335
   -0.1059    0.9122
   -0.3718    0.8001
    0.7453   -0.1821
    0.6167    0.4583
    0.6477   -0.0293
   -0.8714    0.1204
    0.6378   -0.4204
    0.0769    0.3727
m =
    0  1  2  3  1  2  3  4  2  3  4  5
    1  0  1  2  2  1  2  3  3  2  3  4
    2  1  0  1  3  2  1  2  4  3  2  3
    3  2  1  0  4  3  2  1  5  4  3  2
    1  2  3  4  0  1  2  3  1  2  3  4
    2  1  2  3  1  0  1  2  2  1  2  3
    3  2  1  2  2  1  0  1  3  2  1  2
    4  3  2  1  3  2  1  0  4  3  2  1
    2  3  4  5  1  2  3  4  0  1  2  3
```

```
3    2    3    4    2    1    2    3    1    0    1    2
4    3    2    3    3    2    1    2    2    1    0    1
5    4    3    2    4    3    2    1    3    2    1    0
```

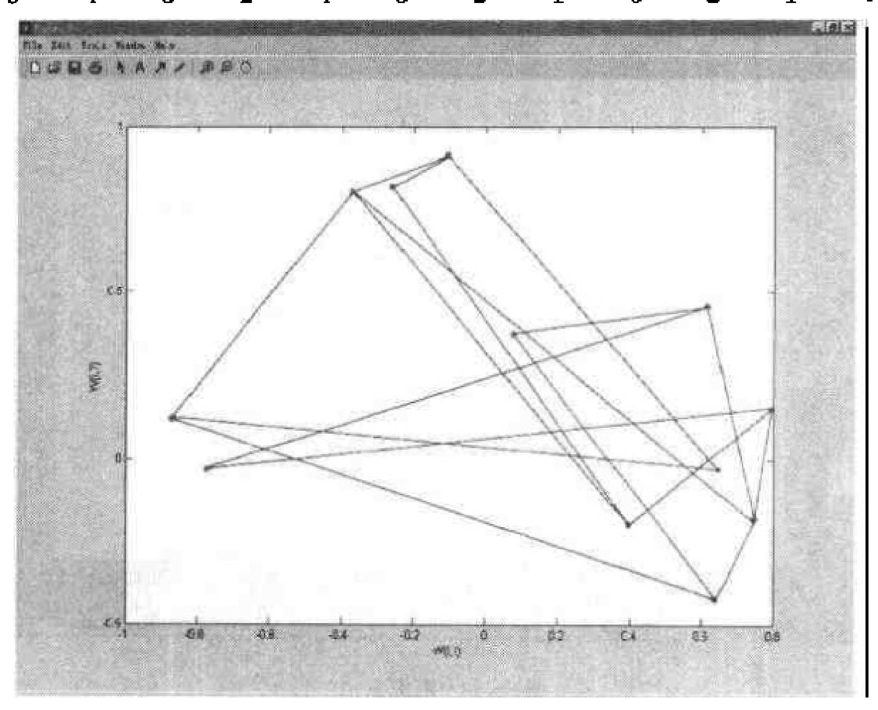


图 10.12　范例 10.25

（2）plotsom()　绘制自组织特征映射网络

语法格式为　plotsom (pos)

　　　　　　plotsom (W, d, nd)

其中参数 pos 表示 s 个 n 维神经元的位置. plotsom(pos) 函数绘图时，用红色的圆点表示神经元的位置，用欧氏距离 1 相连接. plotsom(W, d, nd) 函数中，w 表示权值矩阵，d 表示距离矩阵，nd 表示邻域矩阵，其缺省值为 1. 它连接距离小于 1 的神经元的权值矢量. plotsom() 函数的例可参见例 10.2.

10.5　网络设计实例

10.5.1　竞争学习网络设计实例

下面的例子说明了如何运用竞争算法对输入样本进行分类.

首先，定义输入样本 P 是一组任意产生的测试数据点，它们共有 80 个元素，呈聚类状分布，因此可以应用竞争网络将输入矢量分为不同的类.

例 10.26　输入如下程序，创建测试数据。测试数据点在图 10.13 中用“+”表示。

```
X = [0 1; 0 1];              % 定义聚类中心的范围
clusters = 8;                % 定义聚类个数
points = 10;                 % 定义每个聚类的数据点个数
std_dev = 0.05;              % 定义每个聚类的标准偏差
```

```
P = nngenc (X, dusters, points, std - dev);        %创建聚类的数据点
```

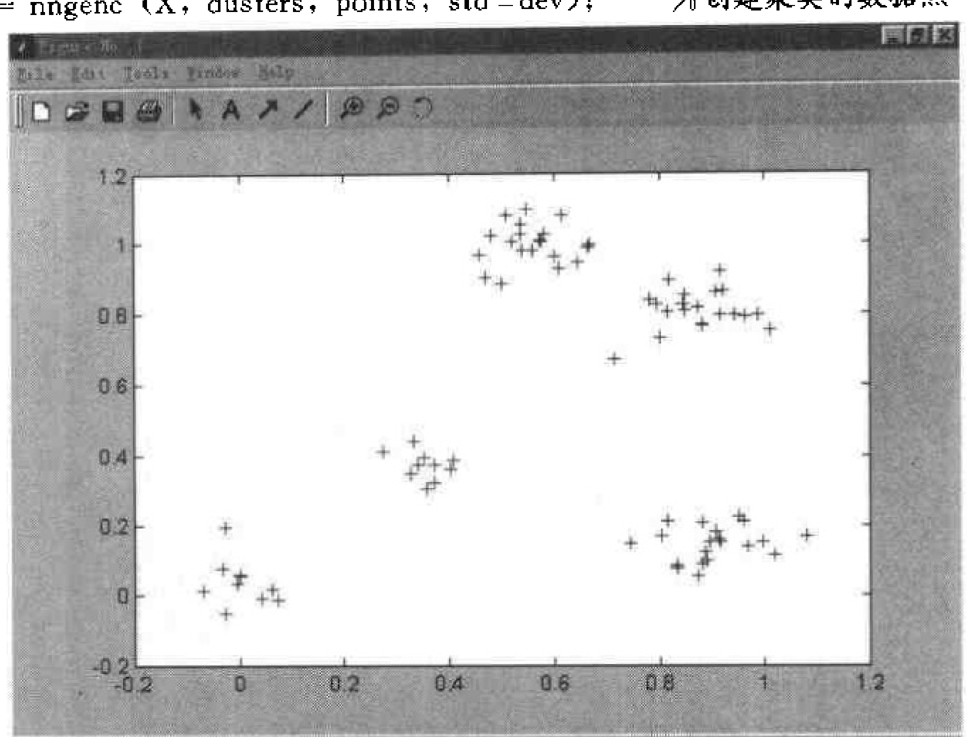


图 10.13 范例 10.26

然后，对网络进行初始化、训练和仿真. 在下面的程序设计中运用了两种不同的方式进行说明.

1）运用 newc() 函数创建竞争层网络，再分别运用网络初始化函数 init()、网络训练函数 train() 和网络仿真函数 sim() 对竞争网络进行初始化、训练和仿真.

例 10.27 输入程序如下:

```
net=newc ( [0 1; 0 1], 8, .1);           %创建一个具有 8 个神经元的竞争层网络
net=init (net);                          %网络初始化
w=net.IW {1};                            %获取网络初始化权值
net.trainParam.epochs=700;               %定义网络最大训练步数
net=train (net, P);                      %网络训练
w=net.IW {1};                            %获取网络训练后权值
plot(P(1,:),P(2,:),'+r', w(:,1),w(:,2),'ob');     %绘制网络输入矢量
                                                    和训练后权值矢量
title ('Input Vectors/ Weight Vectors ');
xlabel ('p (1), w (1)')
ylabel ('p (2), w (2)')
a=0;
p = [0; 0.2];
a—sim (net, p)                      %网络仿真
```

程序运行结果如下

```
a =
```

(4, 1)　　　　1

从图 10.14 中可以看出，权值矢量分布在输入矢量的每个聚类中心.

2) 运用initc() 函数对竞争网络进行初始化，运用 trainc () 函数对竞争层进行训练，运用 simuc() 函数对竞争层进行仿真. 训练过程中将显示频率设定为每 25 步显示一次，最大训练步数为 700 步，学习率为 0.1.

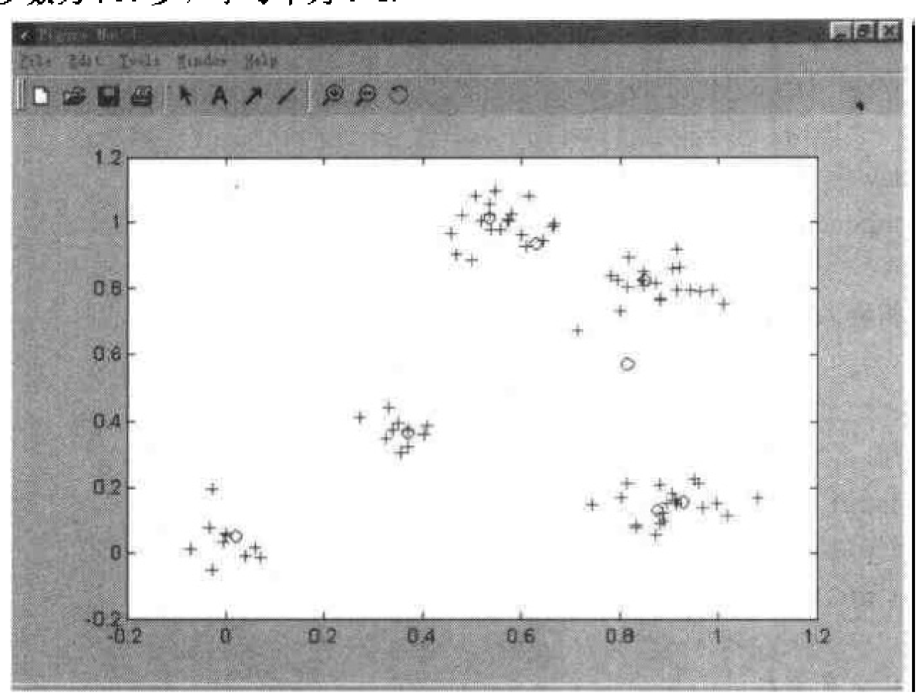


图 10.14　范例 10.27

例 10.28　输入程序如下：

```
w=initc (P, 8);            %竞争层初始化
df=25;
me=700;
lr=0.1;
tp= [df me lr];
w=trainc(w, P, tp);        %竞争层训练
a=0;
p = [0; 0.2];
a=sim(net, p)              %竞争层仿真
```

程序运行结果与方法 1) 的结果一致.

因此，竞争层训练完成后，可以作为分类器用来对输入矢量进行分类. 此时，每个神经元对应于一个不同的聚类.

下面是本例的完整程序：

```
% ==================================
% 利用竞争学习进行模式分类
% ==================================
```

```
% INITC 初始化竞争层
% TRAINC 训练竞争层
% SIMUC 竞争层仿真

% 创建输入样本数据
X = [0 1; 0 1];
clusters = 8;
points = 10;
std_dev = 0.05;
P = nngenc (X, clusters, points, std_dev);

% 绘制输入样本分布
Pause
clc
plot(P(1,:),P(2,:),'+r');
title('Input Vectors');
xlabel('p(1)')
ylabel('p(2)')

% 初始化竞争层
pause
clc
w=initc(P,8);
plot(P(1,:),P(2,:),'+r', w(:,1),w(:,2),'ob');
title('Input Vectors/ Weight Vectors ');
xlabel('p(1),w(1)')
ylabel('p(2),w(2)')

% 训练竞争层
pause
clc
df=25;
me=700;
lr=0.1;
tp= [df me lr];
w=trainc(w, P, tp);

% 竞争层仿真
pause
```

```
clc
p = [0; 0.2];
a=simuc(P, w);
echo off
```

10.5.2　一维空间自组织特征映射网络设计实例

自组织特征映射网络是一种具有侧向联想能力的神经网络，各神经元的连接权值具有一定的分布，最邻近的神经元互相激励，较远的神经元相互抑制，更远一些的则又具有较弱的激励作用.

假设输入矢量为 100 个分布在 0°至 90°的二值单位输入矢量.

例 10.29　输入如下程序，输入矢量图形如图 10.15 所示。

```
angles = 0: 0.5*pi/99: 0.5*pi;
P = [sin (angles); cos (angles)];
```

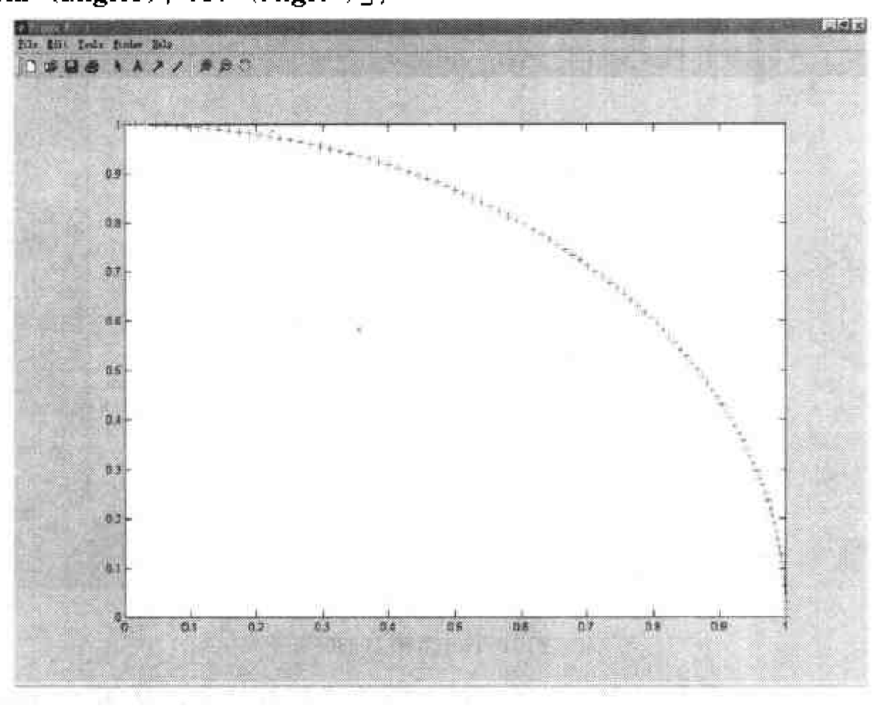


图 10.15　范例 10.29

1）运用 newsom() 函数创建自组织特征映射网络，再分别运用网络初始化函数 init()、网络训练函数 train() 和网络仿真函数 sim() 对竞争网络进行初始化、训练和仿真.

例 10.30　输入如下程序：

```
net=newsom ([0 1; 0 1], [10]);     %创建一个具有 10 个神经元的一维自组织特征映射网络
net=init (net);                     %网络初始化
w=net.IW {1}                        %获取网络初始化权值
net.trainParam.epochs=1000;         %定义网络最大训练步数
net=train (net, P);                 %网络训练
```

```
w=net.IW {1}                                %获取网络训练后权值
plot(P(1,:),P(2,:),'+r', w(:,1),w(:,2),'ob');   %绘制网络输入矢量和训练
                                                后权值矢量
a=0;
a = sim (net, [1; 0])
```

程序运行结果如下

```
a =
    (10, 1)        1
```

由上述程序仿真结果可见，每次只有一个神经元输出 1，因此可以对输入矢量进行分类. 从图 10.16 中可以看出，神经元权值矢量的连接轨迹映射了网络的输入矢量，其权值分布形成了对输入模式的不同响应. 因此，网络学习了输入矢量的拓扑关系.

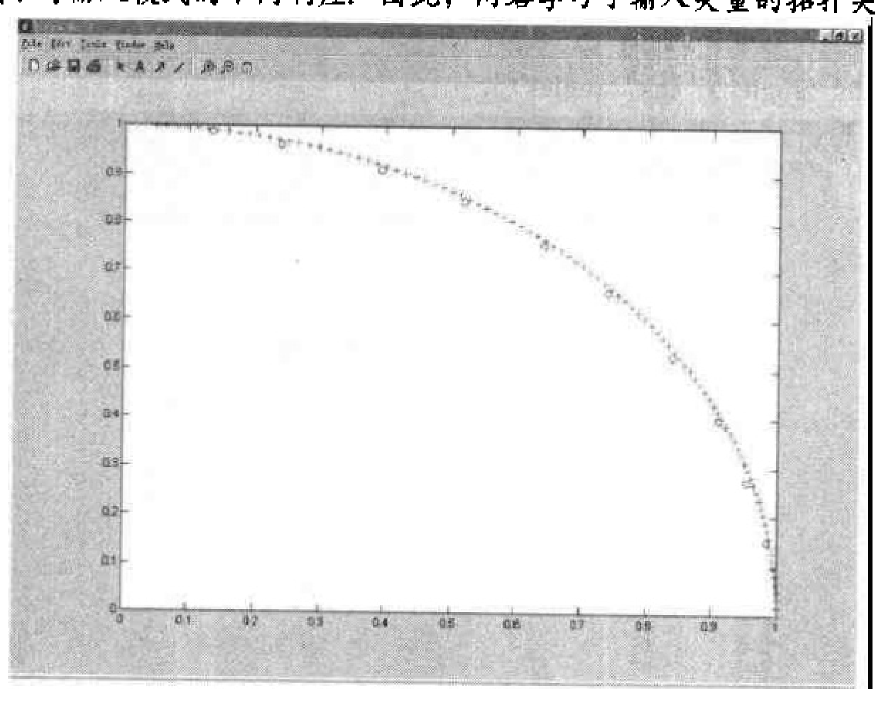


图 10.16 范例 10.30

2) 运用 initsm() 函数对自组织特征映射网络进行初始化，运用 trainsm() 函数和 simusm() 函数对其进行训练和仿真.

首先定义一维网格邻域矩阵和初始权值矩阵.

```
S=10;
M=nbgrid(S);
W=initsm(P,S);
```

然后定义网络训练过程的控制参数：训练过程显示频率定义为每 50 步显示一次，训练迭代的最大次数定义为 1000 步，学习率定义为 0.5.

```
tp= [50 1000 0.5];
W=trainsm(W,M,P,tp)
a=0;
a = simusm([1;0],W,M)
```

程序的运行结果如下

```
a=
     (1, 1)        1.0000
     (2, 1)        0.5000
```

由此可见，上述程序仿真结果只有自组织映射网络输入最大的神经元有输出 1，与获胜神经元相邻的神经元输出 0.5，其余神经元输出为 0.

下面给出本例的完整程序.

```
% ==================================
%  利用自组织特征映射进行模式映射
% ==================================
% INITSM  初始化自组织映射
% TRAINSM  训练自组织映射
% SIMUSM  自组织映射仿真
%创建输入样本数据
angles = 0: 0.5*pi/99: 0.5*pi;
P = [sin(angles); cos(angles)];
plot(P(1,:),P(2,:),'+r')
%初始化自组织映射
pause
clc
S=10;
M=nbgrid(S);
W=initsm(P,S);
%训练自组织映射
pause
clc
tp= [50 1000 0.5];
W=trainsm(W,M,P,tp);
plot(P(1,:),P(2,:),'+r', W(:,1),W(:,2),'ob');
%仿真自组织映射
pause
clc
a=0;
a = simusm ( [1; 0], W, M)
```

第十一章 回归网络

11.1 引 言

MATLAB 的神经网络工具箱包括两种回归网络：Hopfield 网络和 Elman 网络. 在 MATLAB 的工作空间中分别键入 help hopfield 或 help elman 即可获得相关的信息：
键入 help hopfield

```
Hopfield recurrent networks.
  Copyright (c) 1992—1998 by The MathWorks, Inc.
  $Revision: 1.9 $  $Date: 1998/09/03 21: 40: 52 $
  New networks.
    newhop - Create a Hopfield recurrent network.

  Weight functions.
    dotprod - Dot product weight function.

  Net input functions.
    netsum - Sum net input function.

  Transfer functions.
    satlins - Symmetric saturating linear transfer function.

  Demonstrations.
    demohop1 - Two neuron design.
    demohop2 - Unstable equilibria.
    demohop3 - Three neuron design.
    demohop4 - Spurious stable points.
```

键入 help elman

```
Elman recurrent networks.

  New networks.
    newelm - New Elman backpropagation network.

  Using networks.
    sim      - Simulate a neural network.
    init     - Initialize a neural network.
    adapt    - Allow a neural network to adapt.
    train    - Train a neural network.
```

Weight functions.

dotprod - Dot product weight function.

ddotprod- Dot product weight derivative function.

Net input functions.

netsum - Sum net input function.

dnetsum - Sum net input derivative function.

Transfer functions.

purelin - Hard limit transfer function.

tansig - Hyperbolic tangent sigmoid transfer function.

logsig - Log sigmoid transfer function.

dpurelin - Hard limit transfer derivative function.

dtansig - Hyperbolic tangent sigmoid transfer derivative function.

dlogsig - Log sigmoid transfer derivative function.

Performance.

mse - Mean squared error performance function.

msereg - Mean squared error w/reg performance function.

dmse - Mean squared error performance derivatives function.

dmsereg - Mean squared error w/reg performance derivative function.

Learning.

learnbp - Backpropagation learning rule.

learnbpm- Backpropagation learning rule with momentum.

Adaption.

adaptwb - By-weight-and-bias network adaption function.

Training.

traingd - Gradient descent backpropagation.

traingdm- Gradient descent w/momentum backpropagation.

traingda - Gradient descent w/adaptive lr backpropagation.

traingdx - Gradient descent w/momentum & adaptive lr backpropagation.

Initialization.

initlay - Layer-by-layer network initialization function.

initnw - Nguyen-Widrow layer initialization function.

Applications.

appelm1 - Demonstrates Elman recurrent network.

11.2 回归网络

回归网络也称递归网络，在这类网络中，多个神经元互联以组成一个互联神经网络，

如图 11.1 所示. 如图 11.1 中，V_i 表示节点的状态，x_i 为节点的输入（初始）值，x_i' 为收敛后的输出值，$i=1, 2, \cdots, n$. 有些神经元的输出被反馈至同层或前层神经元，因此，信号能够从正向和反向流通. Hopfield 网络和 Elman 网络是回归网络中最有代表性的例子.

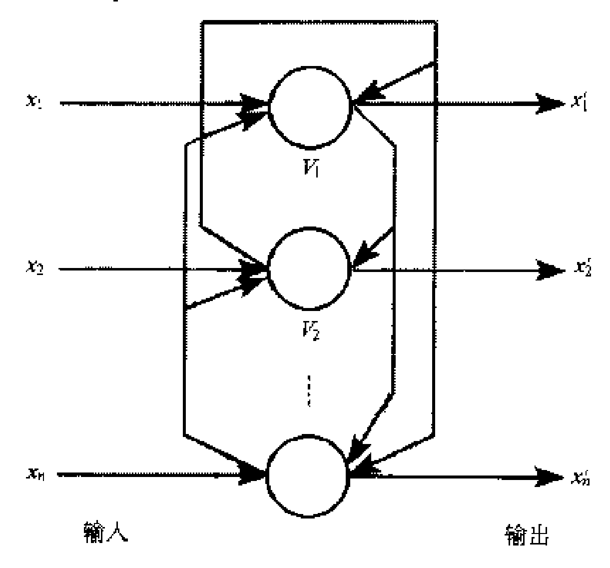


图 11.1　回归网络

11.2.1 Hopfield 网络

Hopfield 神经网络是美国物理学家 J. J. Hopfield 于 1982 年首先提出的. 它主要用于模拟生物神经网络的记忆机理.

Hopfield 神经网络有离散型（DHNN）和连续型（CHNN）两种.

Hopfield 神经网络状态的演变过程是一个非线性动力学系统，可以用一组非线性差分方程（对于离散型的 Hopfield 神经网络）或微分方程（Hopfield 神经网络）来描述. 系统的稳定性可用所谓的"能量函数"（即李雅普诺夫或哈密顿函数）进行分析. 在满足一定条件下，某种"能量函数"的能量在网络运行过程中不断地减少，最后趋于稳定的平衡状态.

对于一个非线性动力学系统，系统的状态从某一初值出发经过演变后可能有如下几种结果：

1）渐近稳定点（吸引子）；

2）极限环；

3）混沌（chaos）；

4）状态发散.

因为人工神经网络的变换函数是一个有界函数，故系统的状态不会产生发散现象. 目前，人工神经网络常利用渐近稳定点来解决某些问题. 例如，如果把系统的稳定点视为一个记忆的话，那么从初态朝这个稳定点的演变过程就是寻找该记忆的过程. 如果把系统的稳定点视为一个能量函数的极小点，而把能量函数视为一个优化问题的目标函数，那么从初态朝这个稳定点的演变过程就是一个求解该优化问题的过程. 由此可见，Hopfield 网络的演变过程是一种计算联想记忆或求解优化问题的过程. 实际上它的解并不需要真

地去计算，而只要构成这种反馈神经网络，适当地设计其连接权和输入就可以达到这个目的.

1. 离散型 Hopfield 神经网络（DHNN）

（1）网络的结构及工作方式

DHNN 是一种单层的、其输入输出为二值的反馈网络，它主要用于联想记忆. DHNN 的结构如图 11.2 所示. 图中定义 $X=[x_1, x_2, \cdots, x_n]^T$ 为网络的状态矢量，其分量是 n 个神经元的输出，仅取 +1 或 −1 二值. $\theta=[\theta_1, \theta_2, \cdots, \theta_n]^T$ 为网络的阈值矢量. $W=[w_{ij}]_{n\times n}$ 为网络的连接权矩阵，其元素 w_{ij} 表示第 i 个神经元到第 j 个神经元的连接权，它为对称矩阵，即 $w_{ij}=w_{ji}$. 若 $w_{ii}=0$，则称其网络为无自反馈的，否则，称其为有自反馈的.

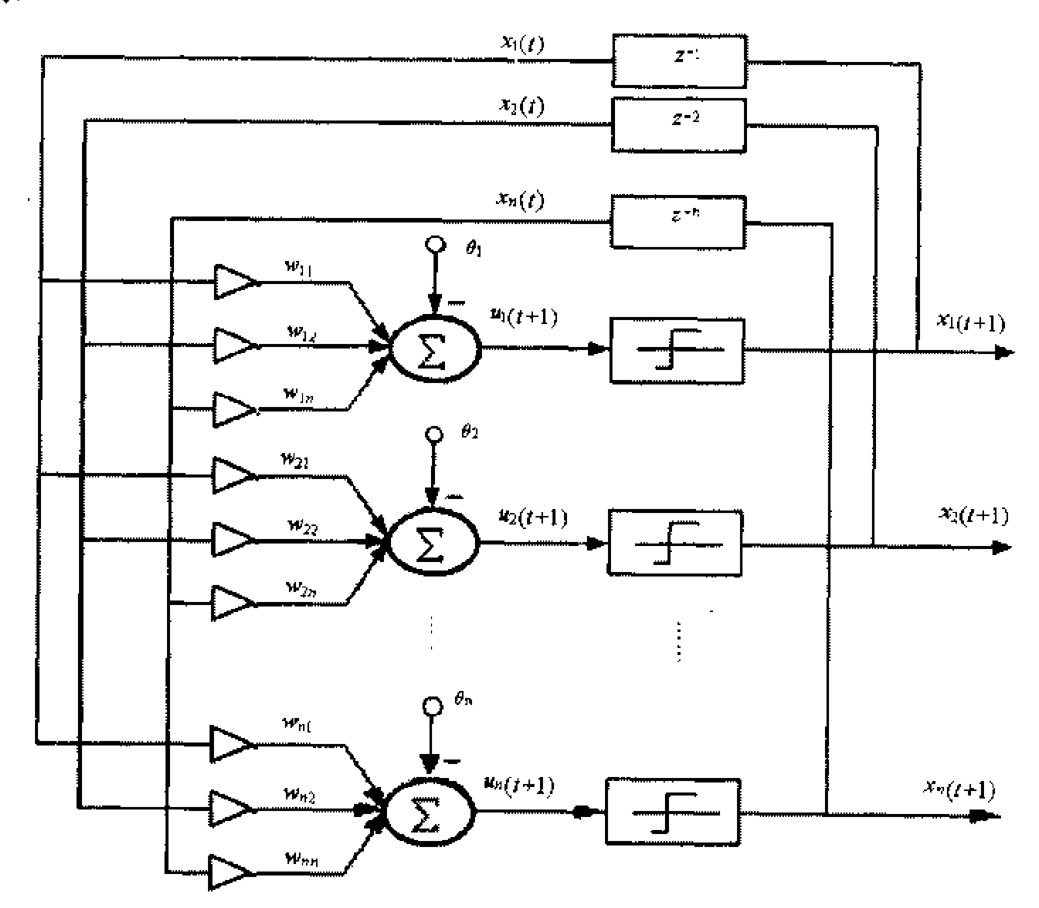


图 11.2 DHNN 结构

DHNN 网络的计算公式如下：

$$u_i(t+1)=\sum_{j=1}^{n} w_{ij}x_j(t)-\theta_i \tag{11.1}$$

$$x_i(t+1)=\operatorname{sgn}[u_i(t+1)] \tag{11.2}$$

式中 sgn(x) 为符号函数

$$\operatorname{sgn}(x)=\begin{cases}1, x\geqslant 0\\ 0, x<0\end{cases}$$

DHNN 主要有以下两种工作方式:

1）串行工作方式

在某一时刻只有一个神经元按照式（11.1）和式（11.2）改变状态，而其余神经元的输出保持不变. 这一变化的神经元可以按照随机方式或预定的顺序来选择. 例如，若选到的神经元为第 i 个，则有

$$x_i(t+1)=\mathrm{sgn}[u_i(t+1)]$$
$$x_j(t+1)=x_j(t),\ (j\neq i)$$

2）并行工作方式

在某一时刻有 N（$1<N\leqslant n$）个神经元按照式（11.1）和式（11.2）改变状态，而其余神经元的输出保持不变. 变化的这一组神经元可以按照随机方式或某种规则来选择. 当 $N=n$ 时，称为全并行方式，即在某一时刻所有的神经元都按式(11.1)和式(11.2)改变状态，亦即

$$x_i(t+1)=\mathrm{sgn}[u_i(t+1)]$$
$$(i=1,2,\cdots,n)$$

若神经网络从某一状态 X（0）开始，经过有限时间 t 后，它的状态不再发生变化，这就是 DHNN 的稳定状态（吸引子）. 用数学公式表示为

$$x_i(t+1)=x_i(t)=\mathrm{sgn}\Big(\sum_{j=1}^{n}w_{ij}x_j(t)-\theta_i\Big)$$
$$(i=1,2,\cdots,n)$$

（2）网络的稳定性分析

如前所述，Hopfield 神经网络的稳定性可以用网络的能量函数进行分析. DHNN 的能量函数定义为

$$E=-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}w_{ij}x_ix_j+\sum_{i=1}^{n}\theta_ix_i$$

写成矩阵形式为

$$E=-\frac{1}{2}X^TWX+X^T\theta$$

由于 x_i，x_j，只能为±1，w_{ij}，θ_i 有界，i，$j=1,2,\cdots,n$.所以能量函数是有界的，即

$$|E|=-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}|w_{ij}||x_i||x_j|+\sum_{i=1}^{n}|\theta_i||x_i|$$
$$=\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}|w_{ij}|+\sum_{i=1}^{n}|\theta_i|$$

若从某一初始状态开始，每次迭代都能满足 $\Delta E=E(t+1)-E(t)\leqslant 0$，即网络的能量单调下降，则网络的状态最后将趋于一个稳定点（有关 DHNN 的稳定性证明，请读者参考其他一些神经网络的理论书籍）.

2. 连续型 Hopfield 神经网络

连续型 Hopfield 神经网络（CHNN）是 J. J. Hopfield 于 1984 年在 DHNN 的基础上提出来的，它的原理与 DHNN 相似. 由于 CHNN 是以模拟量作为网络的输入输出量，各

神经元采用并行方式工作，所以它在信息处理的并行性、联想性、实时性、分布存贮、协同性等方面比 DHNN 更接近于生物神经网络.

(1) 网络模型

图 11.3 是 Hopfield 动态神经元模型，图中电阻 R_{i0} 和电容 C_i 并联，模拟生物神经元的延时特性；电阻 R_{ij} ($j=1, 2, \cdots, n$) 模拟生物神经元之间的突触特性；运算放大器是一个非线性放大器，它模拟生物神经元的非线性特性，其输入、输出关系常用如下的两种单调递增有限非线性函数：

$$\nu_i = \varphi_i(u_i) = \frac{1}{1+e^{-u_i}}$$

$$\nu_i = \varphi_i(u_i) = \tanh(u_i)$$

I_i 为独立的外输入信号.

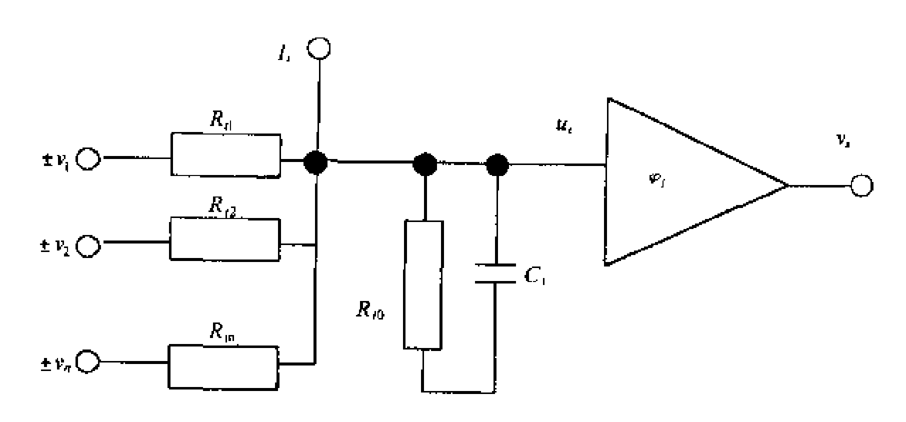


图 11.3　Hopfield 动态神经网络模型

图 11.4 是 CHNN 的结构图. 图中每个神经元可由同相端或反相端输出（图 11.4 未画出反相端）. 当由反相端输出时，它对其他神经元将起抑制作用. 对于每一个神经元而言，自己的输出信号经过其他神经元又反馈到自己，所以 CHNN 是一个连续的非线性动力学系统.

对于第 i 个神经元，放大器的输入、输出关系可用如下的方程来描述：

$$c_i \frac{du_i}{dt} = -\frac{u_i}{R_{i0}} + \sum_{j=1}^{n} \frac{1}{R_{ij}}(\nu_j - u_i) + I_i$$

$$\nu_i = \varphi_i(u_i)$$

上式又可写成

$$\frac{du_i}{dt} = -\frac{u_i}{\tau_i} + \sum_{j=1}^{n} w_{ij}\nu_j + \theta_i$$

$$\frac{1}{\tau_i} = \frac{1}{R_{i0}c_i} + \sum_{j=1}^{n} \frac{1}{R_{ij}c_i}$$

$$w_{ij} = \frac{1}{R_{ij}c_i}$$

$$\theta_i = \frac{I_i}{c_i}$$

对于由 n 个神经元构成的 CHNN，各放大器输入、输出关系可用下式的矩阵方程来描述

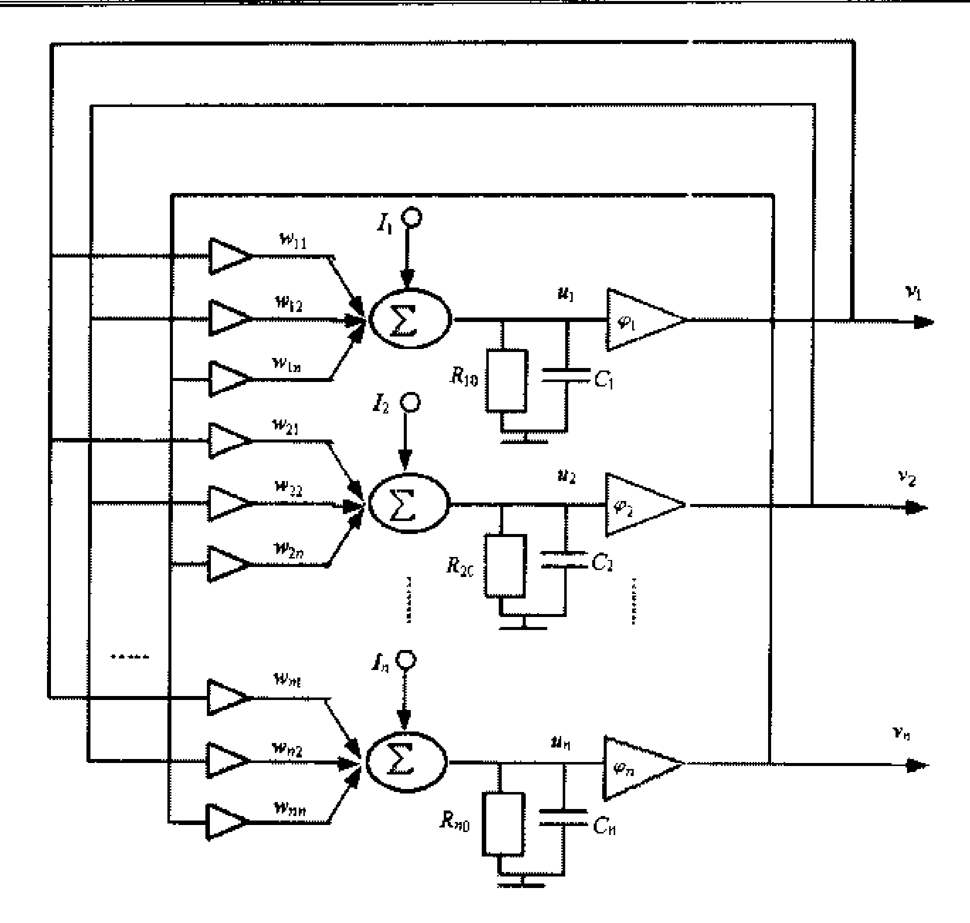


图 11.4　CHNN 结构

$$\dot{u} = -\tau^{-1}u + W\nu + \theta \tag{11.3}$$

$$\nu = \varphi(u)$$

$$\nu = [\nu_1, \nu_2, \cdots, \nu_n]^T$$

$$u = [u_1, u_2, \cdots, u_n]^T$$

$$\varphi(u) = [\varphi_1(u_1), \varphi_2(u_2), \cdots, \varphi_n(u_n)]^T$$

$$\tau = \mathrm{diag}[\tau_1, \tau_2, \cdots, \tau_n]$$

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nn} \end{bmatrix}$$

$$\theta = [\theta_1, \theta_2, \cdots, \theta_n]^T$$

如果令式（11.3）中 $\dot{u}=0$，则有

$$u = \tau W\nu + \tau\theta$$

上式与 DHNN 模型具有相同的形式．如果 φ（·）函数放大器的放大倍数足够大，那么它就变为一个二值硬限幅函数．由此可见，DHNN 可以视为 CHNN 的一种特殊情况．

（2）网络稳定性分析

与 DHNN 一样，网络的稳定性分析是基于网络的能量函数．如前所述，网络的状态

和输出方程为

$$c_i \frac{du_i}{dt} = -\frac{u_i}{R_i} + \sum_{j=1}^{n} w_{ij}\nu_j + I_i$$

$$\nu_i = \varphi_i(u_i) \quad (i = 1,2,\cdots,n)$$

$$\frac{1}{R_i} = \frac{1}{R_{i0}} + \sum_{j=1}^{n} w_{ij}, w_{ij} = \frac{1}{R_{ij}}$$

其能量函数定义为

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\nu_i\nu_j - \sum_{i=1}^{n}\nu_i I_i + \sum_{i=1}^{n}\frac{1}{R}\int_0^{\nu_i}\varphi_i^{-1}(\nu)d\nu$$

式中 $\varphi_i^{-1}(\cdot)$ 是函数 $\nu_i=\varphi_i(u_i)$ 的反函数. 上式第三项表示一种输入状态和输出值关系的能量项. (有关CHNN 的稳定性证明，请读者参考其他一些神经网络的理论书籍.)

11.2.2 Elman 神经网络

图 11.5 给出了Elman 网络的结构. 这种网络具有与MLP 网络相似的多层结构. 在这种网络中，除了普通的隐含层外，还有一个特别的隐含层，有时称为上下文层或状态层，该层从普通隐含层接收反馈信号，上下文层内的神经元输出被前向至隐含层. 如果只有正向连接是适用的，而反馈连接被预定为恒值，那么这些网络可视为普通的前馈网络，而且可以用BP 算法进行训练. 否则，可采用遗传算法.

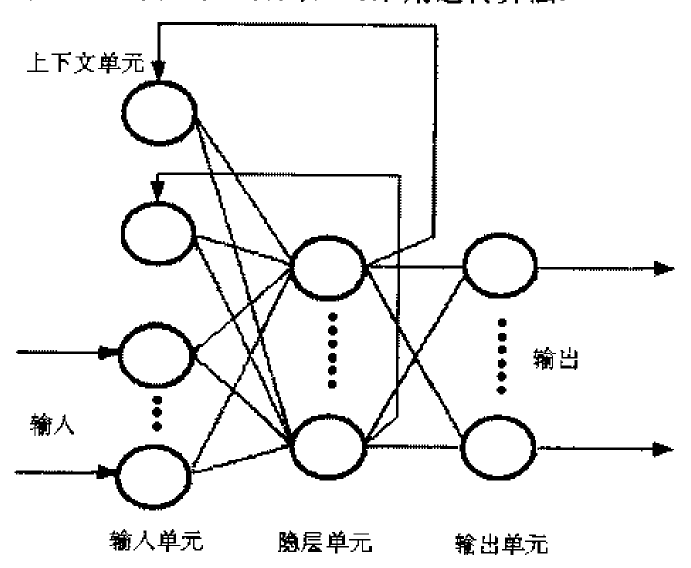


图 11.5　Elman 网络

11.3　回归网络的工具箱函数

关于Hopfiled 和Elman 网络工具箱的部分函数在前几章中已介绍过，这节只介绍新出现的函数.

11.3.1　Hopfield 网络的工具箱函数

1. 网络函数

newhop　设计一个 Hopfield 回归网络

格式：net = newhop (T)

说明：Hopfield 网络经常被应用于模式的联想记忆中. Hopfield 网络仅有一层，其输入用 netsum 函数，权函数用 dotprod 函数，传递函数用 satlins 函数. 层中的神经元有来自它自身的连结权和阈值.

其中

T　目标矢量.

例 11.1　设计一个 Hopfield 网络，其目标矢量 T 为

```
T = [-1 -1 1; 1 -1 1]';
net = newhop (T);
```

下面检验上述这个网络是否稳定在这些点上，

```
Ai = T;
[Y, Pf, Af] = sim (net, 2, [], Ai)
```

运行结果如下：

```
Y =
    -1     1
    -1    -1
     1     1
Pf =
     []
Af =
    -1     1
    -1    -1
     1     1
```

2. 传递函数

satlins　对称饱和线性传递函数

格式：A = satlins(N)

　　info = satlins(code)

说明：对称饱和线性传递函数，传递函数由从网络的输入计算一层的输出. satlins 函数的算法如下：

当 N <= -1 时，satlins (N) = -1;

当 -1 <= N <= 1 时，satlins (N) = N;

当 1 <= N 时，satlins (N) = 1.

其中

N 网络的输入矢量.

satlins (code) 返回信息：

'deriv' 返回导数函数的名称.

'name' 返回全名.

'output' 返回输出范围.

'active' 返回激活输入范围.

例 11.2 绘制出 satlins 函数. 其结果如图 11.6 所示.

```
n = -5: 0.1: 5;
a = satlins (n);
plot (n, a)
```

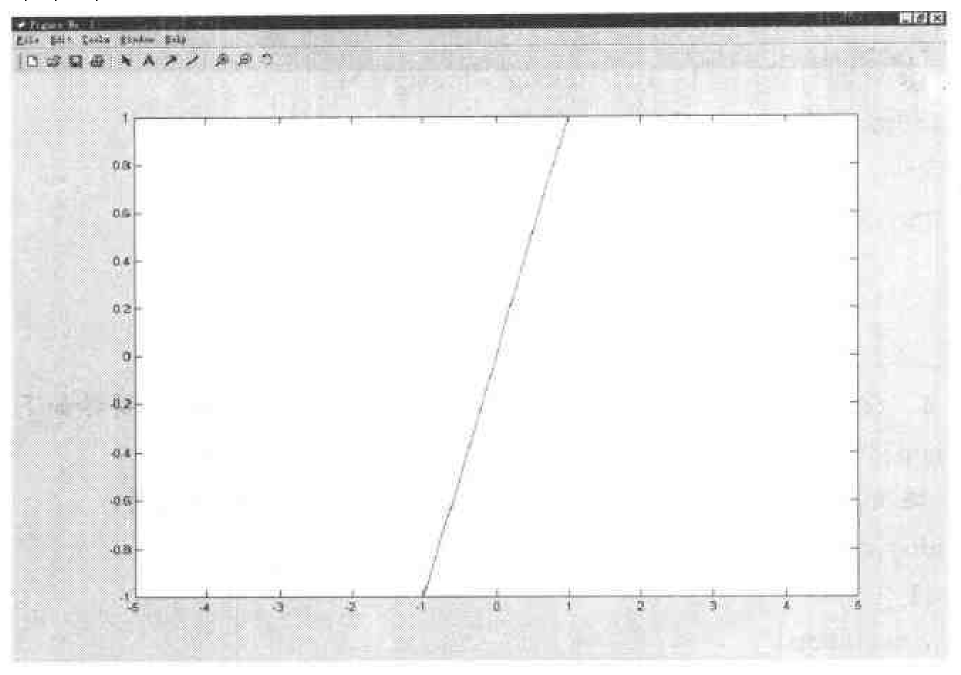


图 11.6 例 11.2 的运行结果

11.3.2 Elman 网络的工具箱函数

newelm (网络函数) 生成一个 Elman 递归网络

格式: net = newelm(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)

说明: Elman 网络由 Nl 层组成，其权函数用 dotprod 函数，输入函数用 netsum 函数，以及用一些特殊的函数作为其传递函数. 每层权和阈值初始化用 initnw.

其中

PR 为输入元素的最大和最小值的阵 (其维数为: R×2)

Si 为第 i 层的神经元个数.

TFi 为第 i 层的传递函数，缺省时= 'tansig'.

BTF 为反向传播网络的训练函数 (BTF 可以是 Traingd, traingdm, traingda, traingdx 等函数)，缺省时= 'traingdx'.

BLF 为反向传播权/阈值学习函数 (BLF 可以是 learngd, 或 learngdm)，缺省时= 'learngdm'.

PF 为性能分析函数（PF 可以是 mse 或 msereg），缺省时= 'mse'.

例 11.3　设定布尔输入 P，和另一个序列：

```
P = round(rand(1,20));
T = [0 (P(1:end-1)+P(2:end) == 2)];
```

下面用 Elman 网络识别(P(1:end-1)+P(2:end) == 2)的发生，首先进行如下转换：

```
Pseq = con2seq(P);
Tseq = con2seq(T);
```

设计一 Elman 网络，其输入范围为(0,1)，网络隐含层有五个神经元和一个输出神经元.

```
net = newelm([0 1],[10 1],{'tansig','logsig'});
```

设目标均方误差为 0.1，对网络进行训练：

```
net = train(net,Pseq,Tseq);
Y = sim(net,Pseq)
```

11.4　应用举例

例 11.4　设计一个含有两个神经元的 Hopfield 网络，其原程序及解释如下（其程序名为 demohop1，程序运行结果如图 11.7～11.14 所示）：

```
%这是一个运用 playshow.m 和 makeshow.m 建立出的 slideshow 文件.
function slide=demohop1
if nargout<1,
  playshow demohop1
else

%画出第一幅图 (Slide 1)
  slide (1) .code= {
    'slideData=nnslides (''start'',slideData,''A Two Neuron Hopfield Network'');' };
  slide (1) .text= {
    'NEWHOP --- Creates a Hopfield network.',
    'SIM —— Simulates a neural network.',
    '',
    'A Hopfield network consisting of two neurons is designed with two stable equilibrium points and simulated using the above functions.'};

%画出第二幅图 (Slide 2)
  slide (2) .code= {
    'slideData=nnslides (''blank'', slideData);',
    'T = [+1 -1; -1 +1];' };
  slide (2) .text= {
```

```
  'We would like to obtain a Hopfield network that has the two stable points defined
by the two target (column) vectors in T.',
  '>> T = [+1 -1;',
  ' -1 +1];',
  ''};
%画出第三幅图 (Slide 3)
 slide (3) .code= {
  'slideData=nnslides (''axes'', slideData);',
  'cla;',
  'plot (T (1,:), T (2,:),''r*'')',
  'axis ( [-1.1 1.1 -1.1 1.1])',
  'title (''Hopfield Network State Space'')',
  'xlabel (''a (1)'');',
  'ylabel (''a (2)'');' };
 slide (3) .text= {
  'Here is a plot where the stable points are shown at the corners. All possible states
of the 2-neuron Hopfield network are contained within the plots boundaries.',
  '>> plot (T (1,:), T (2,:),''r*'')',
  '>> axis ( [-1.1 1.1 -1.1 1.1])',
  '>> title (''Hopfield Network State Space'')',
  '>> xlabel (''a (1)''); ylabel (''a (2)'');',
  ''};
%画出第四幅图 (Slide 4)
 slide (4) .code= {
  ' net=newhop (T);' };
 slide (4) .text= {
  'The function NEWHOP creates Hopfield networks given the stable points T.',
  '>> net=newhop (T);',
  ''};
%画出第五幅图 (Slide 5)
 slide (5) .code= {
  ' [Y, Pf, Af] = sim (net, 2, [], T);' };
 slide (5) .text= {
  'First we check that the target vectors are indeed stable. We check this by giving
the target vectors to the Hopfield network. It should return the two targets unchanged,
and indeed it does:',
  '>> [Y, Pf, Af] = sim (net, 2, [], T); Y',
  'Y =',
```

```
'         1        -1',
'        -1         1',
''};
```

%画出第六幅图 (Slide 6)

```
slide (6) .code= {
'cla;',
'plot (T (1,:), T (2,:),''r*'')',
'axis ( [-1.1 1.1 -1.1 1.1])',
'a = {rands (2, 1)};',
' [y, Pf, Af] = sim (net, {1 20}, {}, a);' };
slide (6) .text= {
'Here we define a random starting point and simulate the Hopfield network for20 steps. It should reach one of its stable points.',
'',
'>> a = {rands (2, 1)};',
'>> [y, Pf, Af] = sim (net, {1 20}, {}, a);',
''};
```

%画出第七幅图 (Slide 7)

```
slide (7) .code= {
'cla;',
'plot (T (1,:), T (2,:),''r*'')',
'axis ( [-1.1 1.1 -1.1 1.1])',
'record= [cell2mat (a) cell2mat (y)];',
'start=cell2mat (a);',
'hold on',
'plot (start (1, 1), start (2, 1),''bx'', record (1,:), record (2,:))',
'' };
slide (7) .text= {
'We can make a plot of the Hopfield networks activity.',
'>> record= [cell2mat (a) cell2mat (y)];',
'>> start=cell2mat (a);',
'>> hold on',
'>> plot (start (1, 1), start (2, 1),''bx'', record (1,:), record (2,:))',
'',
'Sure enough, the network ends up in either the upper-left or lower right corners of the plot.',
''};
```

%画出第八幅图 (Slide 8)

```
slide (8) .code= {
  'color = ''rgbmy'';',
  'for i=1: 25',
  '    a = {rands (2, 1)}; [y, Pf, Af] = sim (net, {1 20}, {}, a);',
  '   record= [cell2mat (a) cell2mat (y)]; start=cell2mat (a);',
  '
  plot (start (1, 1), start (2, 1),''kx'', record (1,:), record (2,:), color (rem
(i, 5) +1))',
  '   drawnow',
  'end' };
slide (8) .text= {
  'We repeat the simulation for 25 more initial conditions.',
  '',
  'Note that if the Hopfield network starts out closer to the upper-left, it will go to
the upper-left, and vise versa. This ability to find the closest memory to an initial in-
put is what makes the Hopfield network useful.',
  ''};
end
```

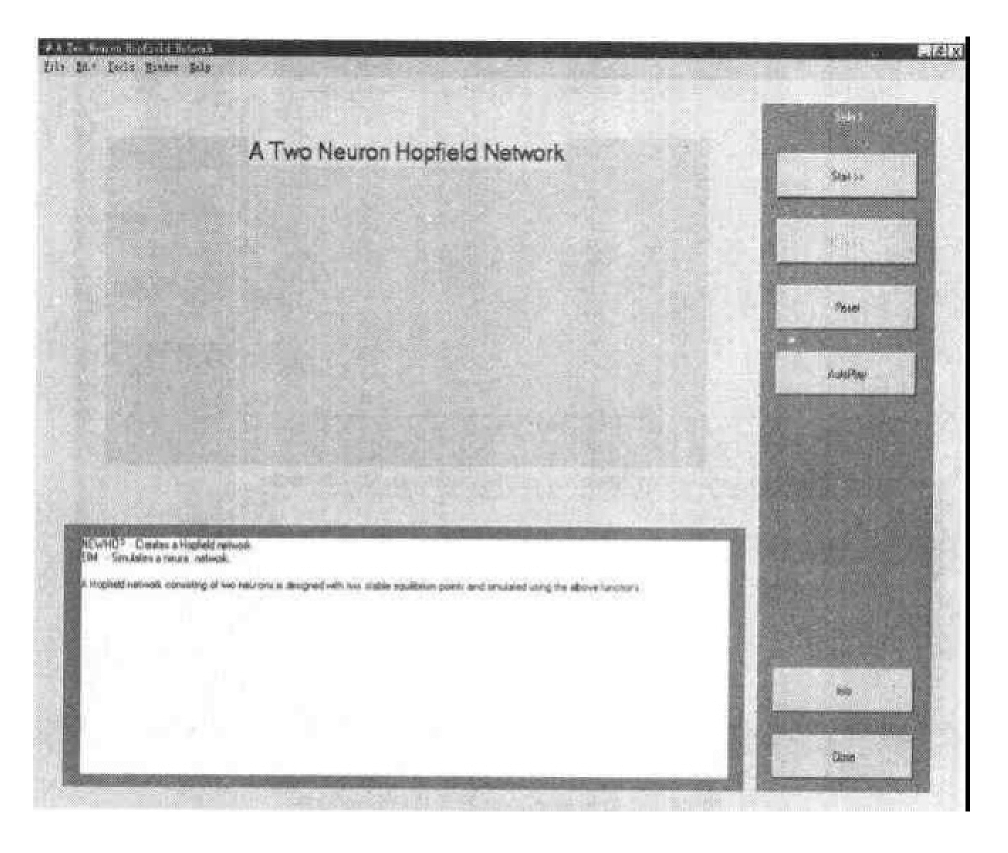


图 11.7 运行程序的菜单选择窗口

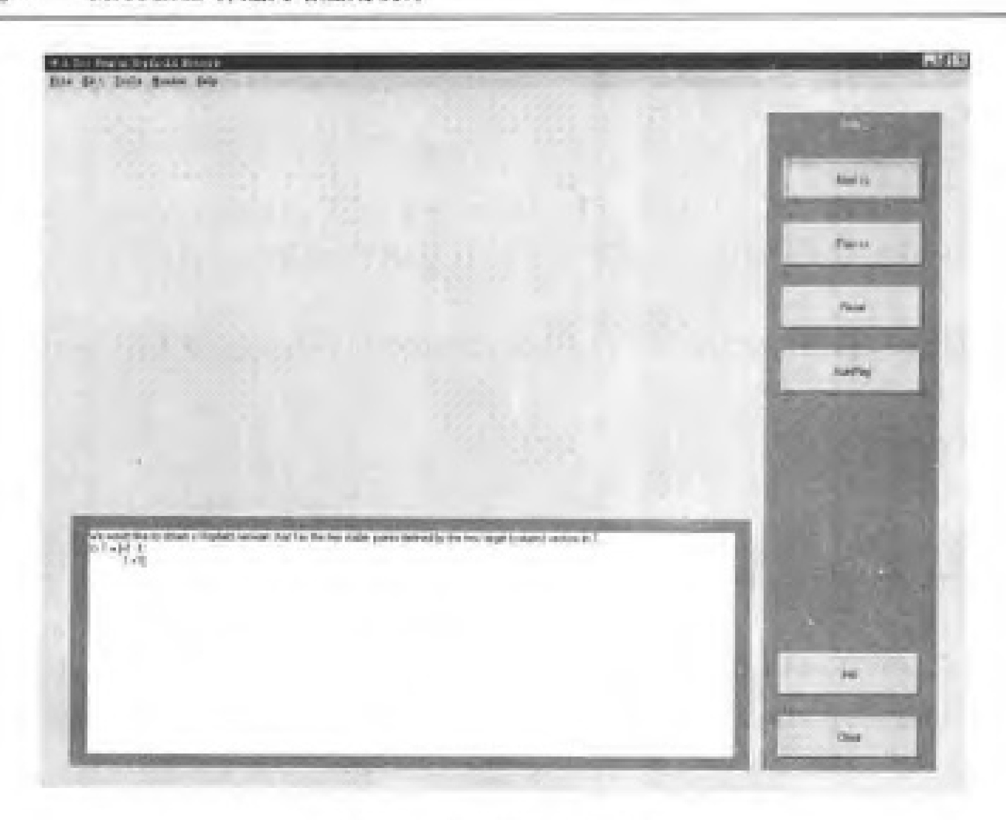


图 11.8　给出目标矢量的定义

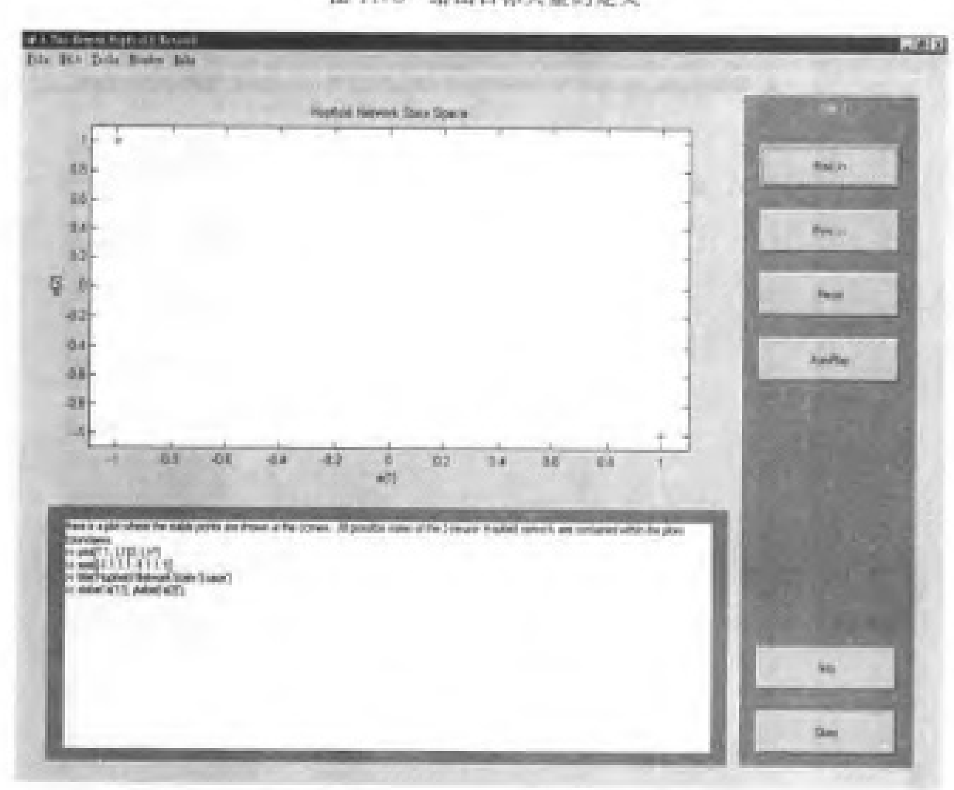


图 11.9　给出两个稳定点（目标）的位置

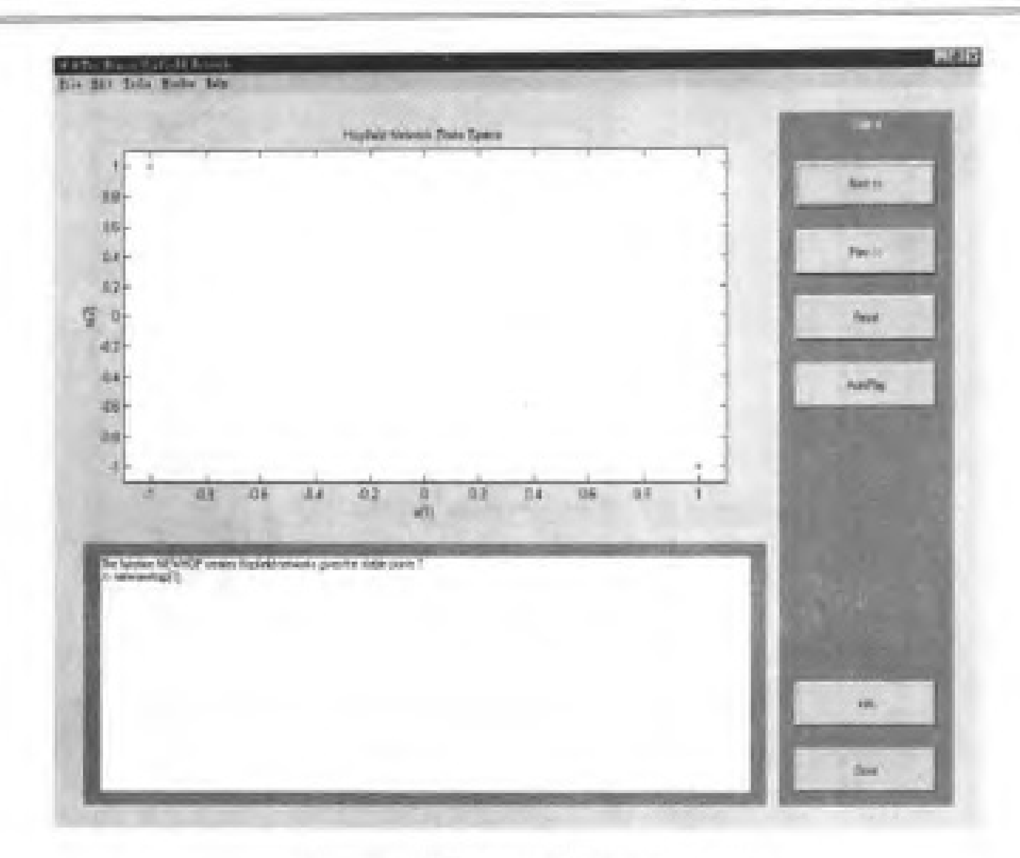


图 11.10　用稳定点（T）设计 Hopfield 网络

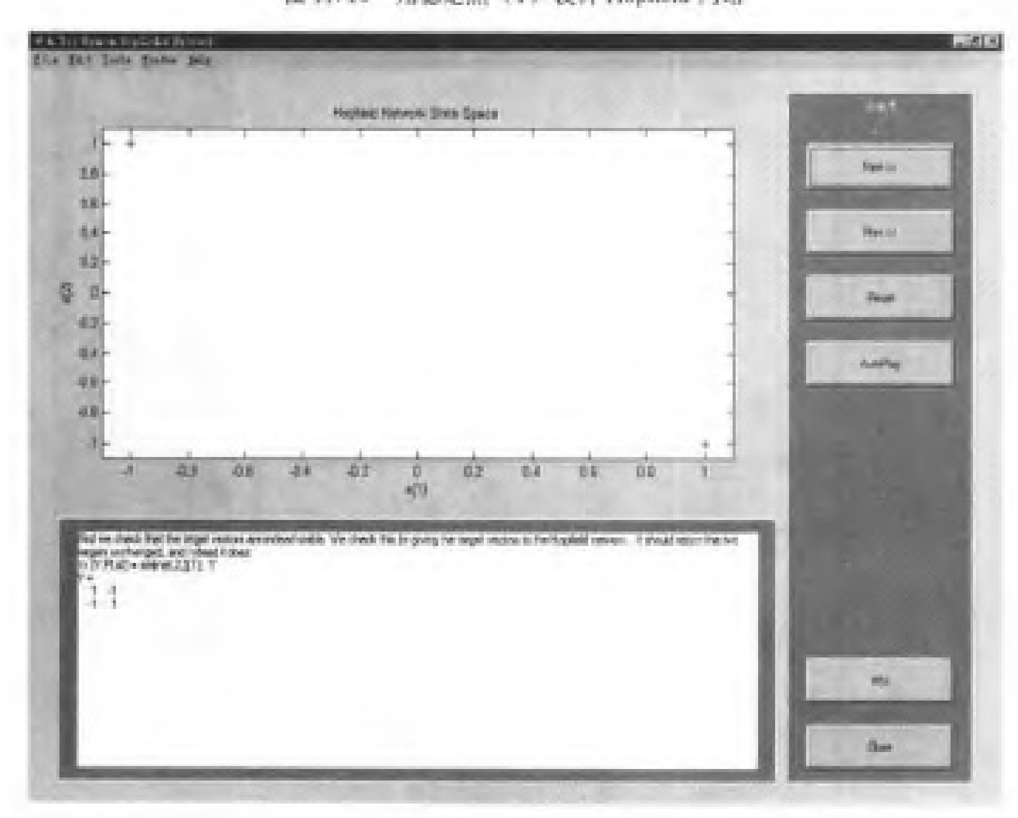


图 11.11　检验目标矢量是否稳定

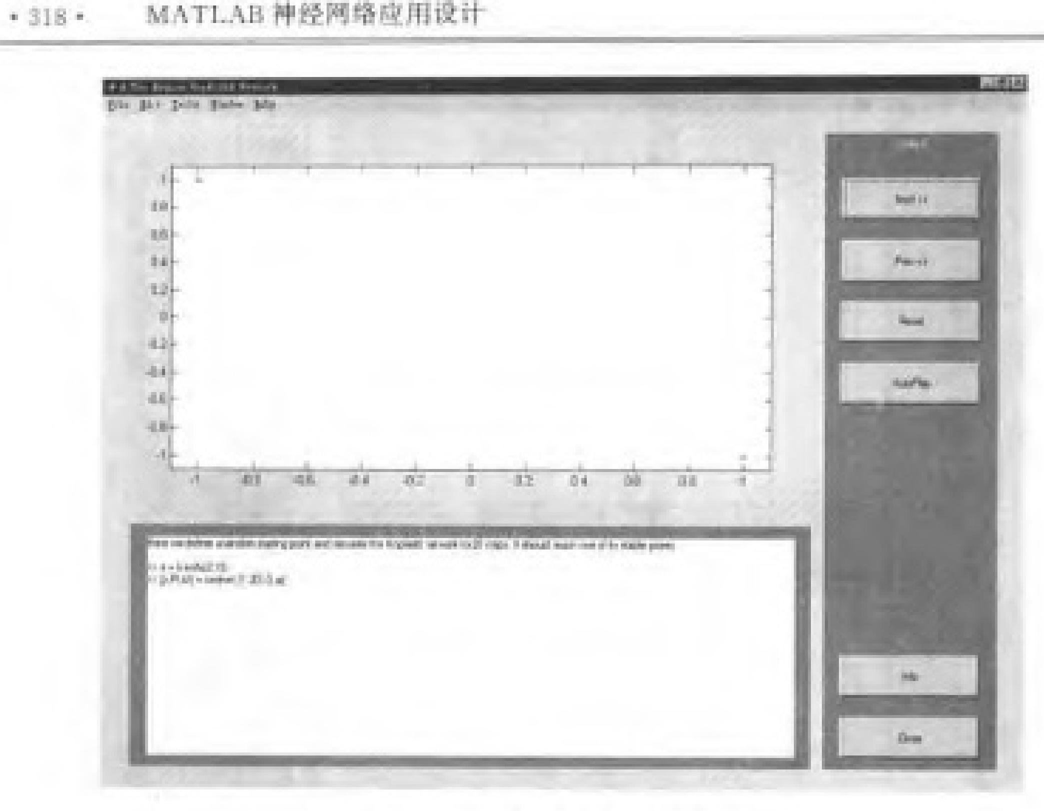


图 11.12　定义随机起始点进行仿真

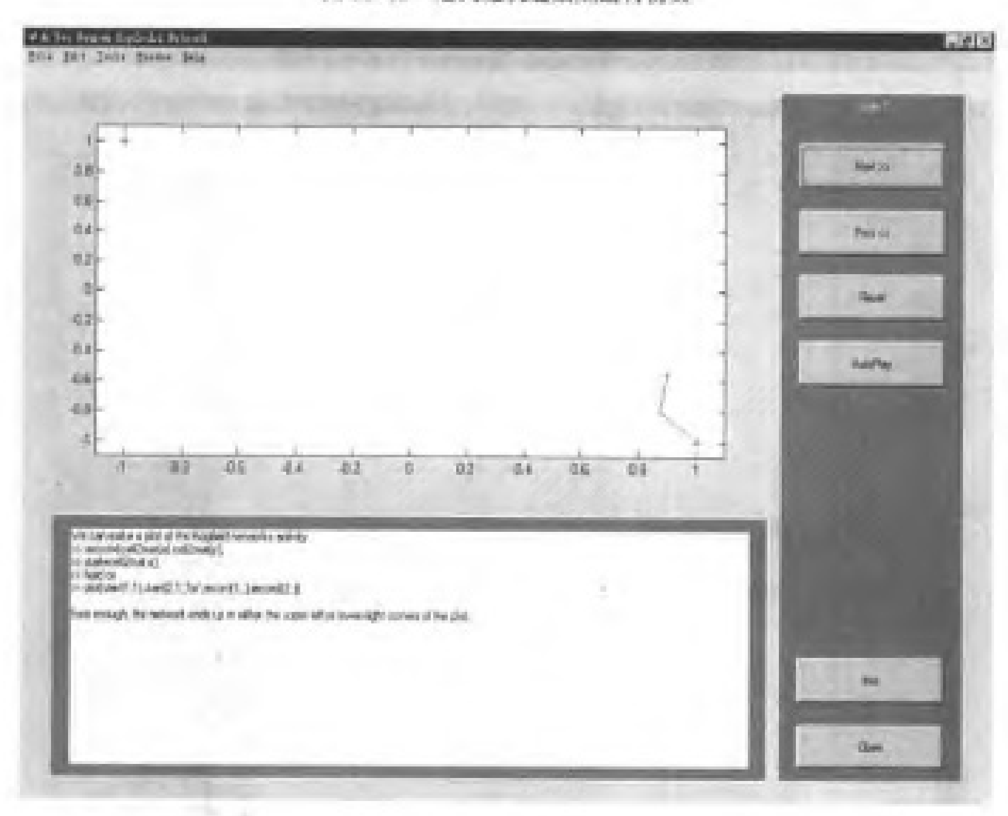


图 11.13　运行结果

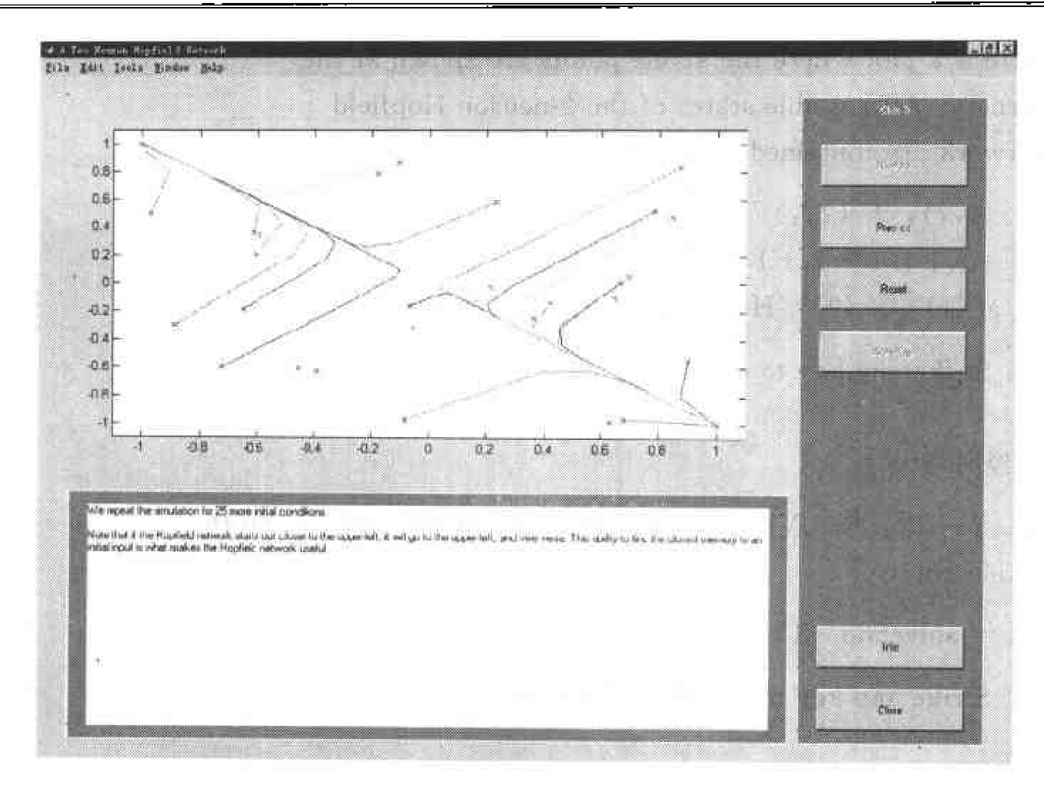


图 11.14 在 25 个初始条件下的仿真结果

为有助于读者对照理解，现给出例 11.4 在 MATLAB5.1 版本下的程序：

```
clf;
figure (gcf)
colordef (gcf,'none')
setfsize (400, 400);
echo on
clc

%   SOLVEHOP —— 设计 Hopfield 网络.
%   SIMUHOP —— 对 Hopfield 网络进行仿真.

%   设计由两个神经元组成的 Hopfield.

%   A Hopfield network consisting of two neurons is designed
%   with two stable equilibrium points and simulated using
%   the above functions.

pause % Strike any key to define the problem...
clc
%   问题定义
%   We would like to obtain a Hopfield network that has the two
%   stable points define by the two target (column) vectors in T.

  T = [+1 -1;
       -1 +1];
```

```
%   Here is a plot where the stable points are shown at the
%   corners. All possible states of the 2-neuron Hopfield
%   network are contained within the plots boundaries.

plot (T (1,:), T (2,:),'r*')
axis ( [-1.1 1.1 -1.1 1.1])
alabel ('a (1)','a (2)','Hopfield Network State Space')

pause % Strike any key to design the network...
clc
%   神经网络设计

%   The function SOLVEHOP designs Hopfield networks given the
%   stable points T.

[W, b] = solvehop (T)

pause % Strike any key to simulate the network...
clc
%   神经网络仿真

%   First we check that the target vectors are indeed stable.

%   We check this by given the target vectors to the Hopfield
%   network. It should return the targets unchanged.

A = simuhop (T, W, b)

%   Sure enough, these are the target vectors [+1; -1] and
%   [-1; +1] .

pause % Strike any key to simulate the network again...
clc
%神经网络仿真
%   Here we define random starting point and simulate the Hopfield
%   network for 50 steps. It should reach one of its stable points.

a = rands (2, 1);
[a, aa] = simuhop (a, W, b, 50);

%   We can make a plot of the Hopfield networks activity.

hold on
plot (aa (1, 1), aa (2, 1),'wx', aa (1,:), aa (2,:))

%   Sure enough, the network ends up in either the upper-left or
%   lower right corners of the plot.

pause % Strike any key to simulate the network some more...
```

```
clc
%神经网络仿真
%   The following code simulates the Hopfield for 25 more initial
%   conditions.
hold on
color = 'rgbmy';
for i=1: 25
    a = rands (2, 1);
    [a, aa] = simuhop (a, W, b, 20);
    plot (aa (1, 1), aa (2, 1),'wx', aa (1,:), aa (2,:), color (rem (i, 5) +1))
    drawnow
end
%   Note that if the Hopfield network starts out closer to the
%   upper-left, it will go to the upper-left, and vise versa.
%   This ability to find the closest memory to an initial input
%   is what makes the Hopfield network useful.
echo off
disp ('End of DEMOHOP1')
```

例 11.5　设计含有三个神经元的网络，其原程序及解释如下（其程序名为 demohop3，程序运行结果如图 11.15～11.22 所示）：

```
%这是一个运用 playshow.m 和 makeshow.m 建立出的 slideshow 文件.
function slide=demohop3
if nargout<1,
  playshow demohop3
else
%画出第一幅图 (Slide 1)
  slide (1) .code= {
   'slideData = nnslides (''start'', slideData'',''A Three Neuron Hopfield Network'');',
   '' };
  slide (1) .text= {
   'NEWHOP - Creates a Hopfield network.',
   'SIM - Simulates a neural network.',
   '',
   'A Hopfield network is designed with target stable points. The behavior of the Hopfield network for different initial conditions is studied.',
   ''};
```

```
%画出第二幅图 (Slide 2)
  slide (2) .code= {
   'slideData=nnslides (''blank'', slideData);',
   'cla;',
' T = [+1 +1; -1 +1;      -1 -1];',
   '' };
  slide (2) .text= {
   ' We would like to obtain a Hopfield network that has the two stable points defined
by the two target (column) vectors in T.',
   '>> T = [+1 +1;',
   '              -1 +1;',
   '              -1 -1];',
   '',
   '',
   ''};
%画出第三幅图 (Slide 3)
  slide (3) .code= {
   'slideData=nnslides (''axes'', slideData);',
   'cla;',
   'axis ( [-1 1 -1 1 -1 1])',
   'set (gca,''box'',''on''); axis manual; hold on;',
   'plot3 (T (1,:), T (2,:), T (3,:),''r *'')',
   'title (''Hopfield Network State Space'')',
   'xlabel (''a (1)'');',
   'ylabel (''a (2)'');',
   'zlabel (''a (3)'');',
   'view ( [37.5 30]);' };
  slide (3) .text= {
   'Here is a plot where the stable points are shown at the corners. All possible states
of the 2-neuron Hopfield network are contained within the plots boundaries.',
   '>> plot3 (T (1,:), T (2,:), T (3,:),''r *'')',
   '>> axis ( [-1 1 -1 1 -1 1]); axis manual; hold on;',
   '>> title (''Hopfield Network State Space'')',
   '>> xlabel (''a (1)''); ylabel (''a (2)''); zlabel (''a (3)'');',
   '>> set (gca,''box'',''on''); view ( [37.5 30]);'};
%画出第四幅图 (Slide 4)
  slide (4) .code= {
   '',
```

```
  ' net=newhop (T);' };
slide (4) .text= {
  'The function NEWHOP creates Hopfield networks given the stable points T.',
  '>> net=newhop (T);',
  ''};
```

%画出第五幅图 (Slide 5)

```
slide(5).code={
  'cla;',
  'a = {rands(3,1)};',
  '[y,Pf,Af] = sim(net,{1 10},{},a);' };
slide(5).text={
  'Here we define a random starting point and simulate the Hopfield network for 50 steps. It should reach one of its stable points.',
  '',
  '>> a = {rands(3,1)};',
  '>> [y, Pf, Af] = sim (net, {1 10}, {}, a);',
  ''};
```

%画出第六幅图 (Slide 6)

```
slide (6) .code= {
  'cla;',
  'plot3(T(1,:),T(2,:),T(3,:),''r*'')',
  'record=[cell2mat(a) cell2mat(y)];',
  'start=cell2mat(a);',
  'plot3(start(1,1),start(2,1),start(3,1),''bx'',record(1,:),record(2,:),record(3,:))' };
  slide(6).text={
  'We can make a plot of the Hopfield networks activity.',
  '>> record=[cell2mat(a) cell2mat(y)];',
  '>> start=cell2mat(a);',
  '>> hold on',
  '>>
plot3(start(1,1),start(2,1),start(3,1),''bx'',record(1,:),record(2,:),record(3,:))',
  '',
  'Sure enough, the network ends up at a designed stable point in the corner.'};
```

%画出第七幅图 (Slide 7)

```
slide (7) .code= {
  'slideData=nnslides(''lock'',slideData);',
```

```
    'cla;',
    'plot3(T(1,:),T(2,:),T(3,:),''r*'')',
    'color = ''rgbmy'';',
    'for i=1:25',
    'a = {rands(3,1)}; [y,Pf,Af] = sim(net,{1 10},{},a);',
    'record=[cell2mat(a) cell2mat(y)]; start=cell2mat(a);',
    'plot3(start(1,1),start(2,1),start(3,1),''kx'',record(1,:),record(2,:),record
(3,:),color(rem(i,5)+1))',
    '    drawnow',
    'end',
    'slideData=nnslides(''unlock'',slideData);' };
  slide(7).text={
    'We repeat the simulation for 25 more randomly generated initial conditions.'};
%画出第八幅图 (Slide 8)
  slide(8).code={
    'cla;',
    'plot3(T(1,:),T(2,:),T(3,:),''r*'')',
    'P = [ 1.0   -1.0   -0.5   1.00   1.00   0.0; 0.0   0.0   0.0   0.00
0.00 -0.0;-1.0   1.0   0.5   -1.01   -1.00   0.0];',
    'color = ''rgbmy'';',
    'for i=1:6',
    'a = {P(:,i)};      [y,Pf,Af] = sim(net,{1 10},{},a);',
    'record=[cell2mat(a) cell2mat(y)]; start=cell2mat(a);',
    'plot3(start(1,1),start(2,1),start(3,1),''kx'',record(1,:),record(2,:),record
(3,:),color(rem(i,5)+1))',
    '    drawnow',
    'end' };
    slide(8).text={
    ' Now we simulate the Hopfield for the following initial conditions, each a column
vector of P:',
    '>> P = [ 1.0   -1.0   -0.5   1.00   1.00   0.0;',
    '             0.0    0.0    0.0   0.00   0.01   -0.0;',
    '            -1.0    1.0    0.5   -1.01  -1.00   0.0]',
    'These points were exactly between the two target stable points. The result is that
they all move into the center of the state space, where an undesired stable point ex-
ists.',
    '',
    ''};
end
```

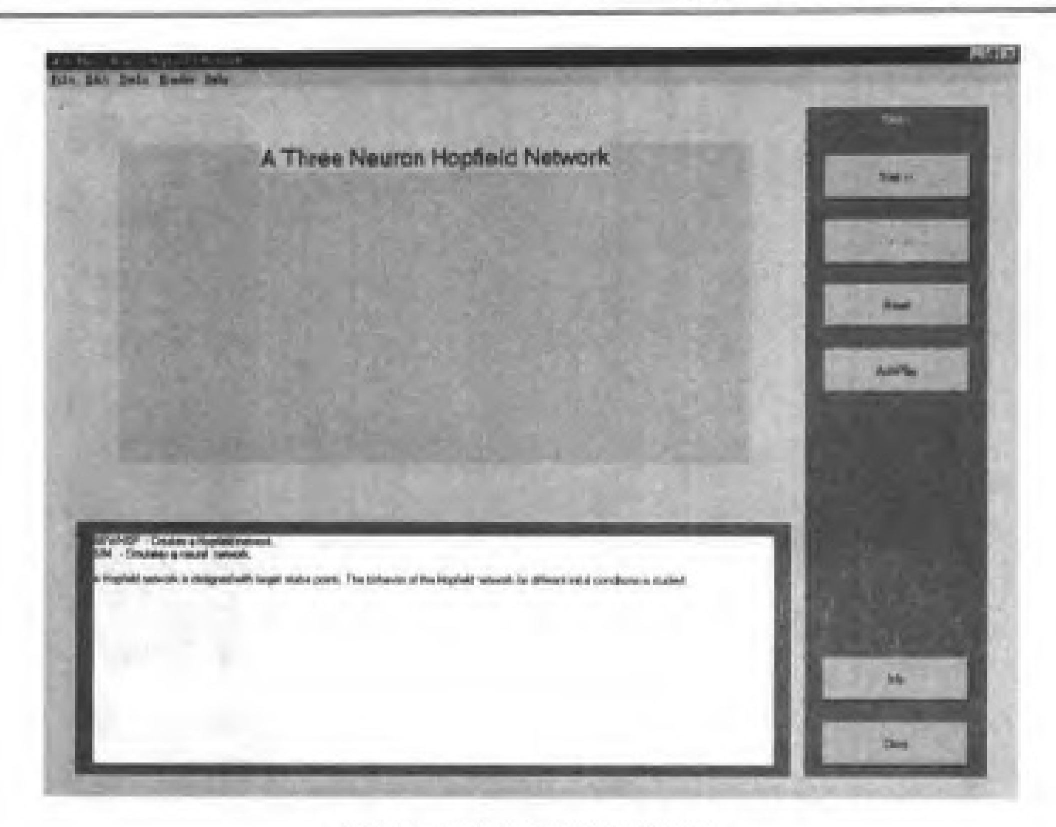


图 11.15 运行程序的菜单选择窗口

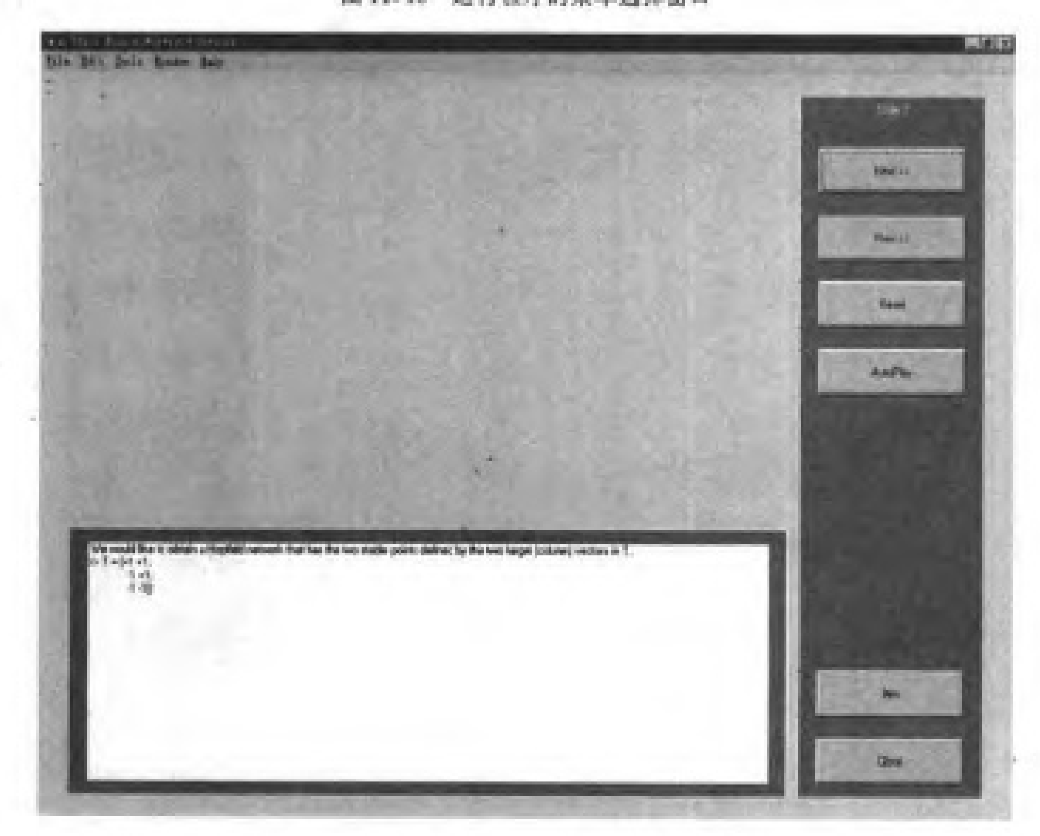


图 11.16 给出目标矢量的定义

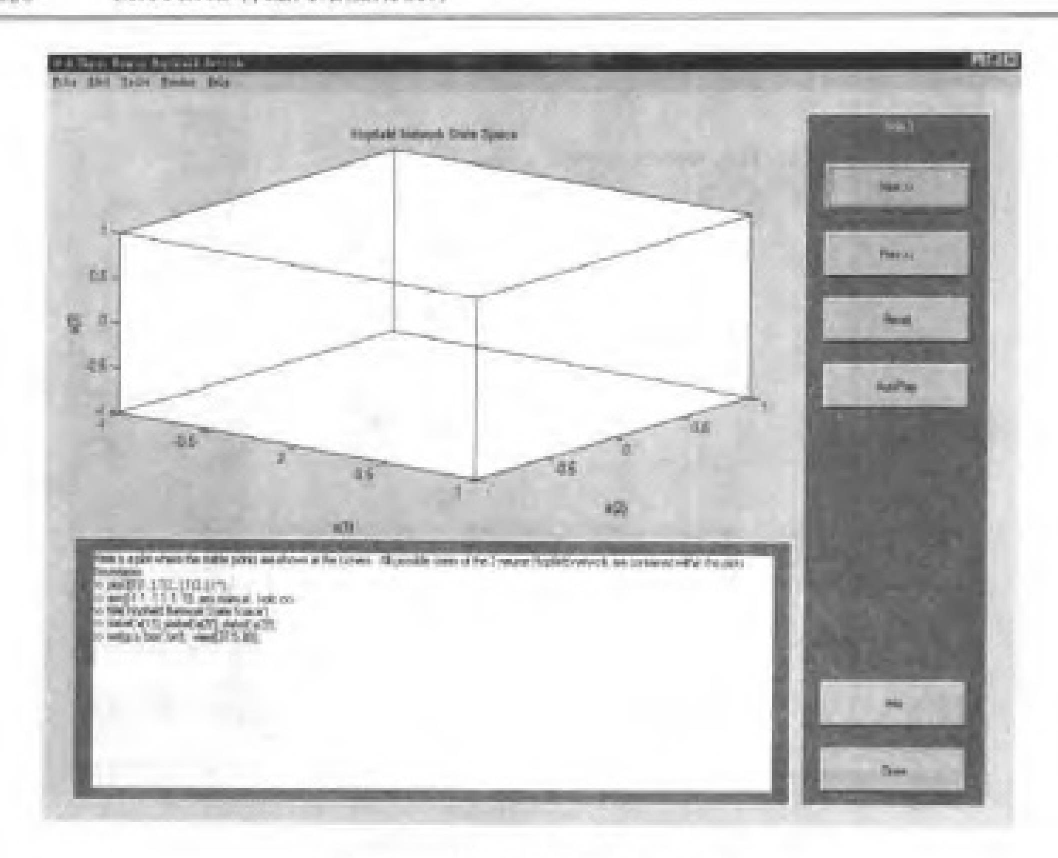


图 11.17　给出稳定点（目标）的位置

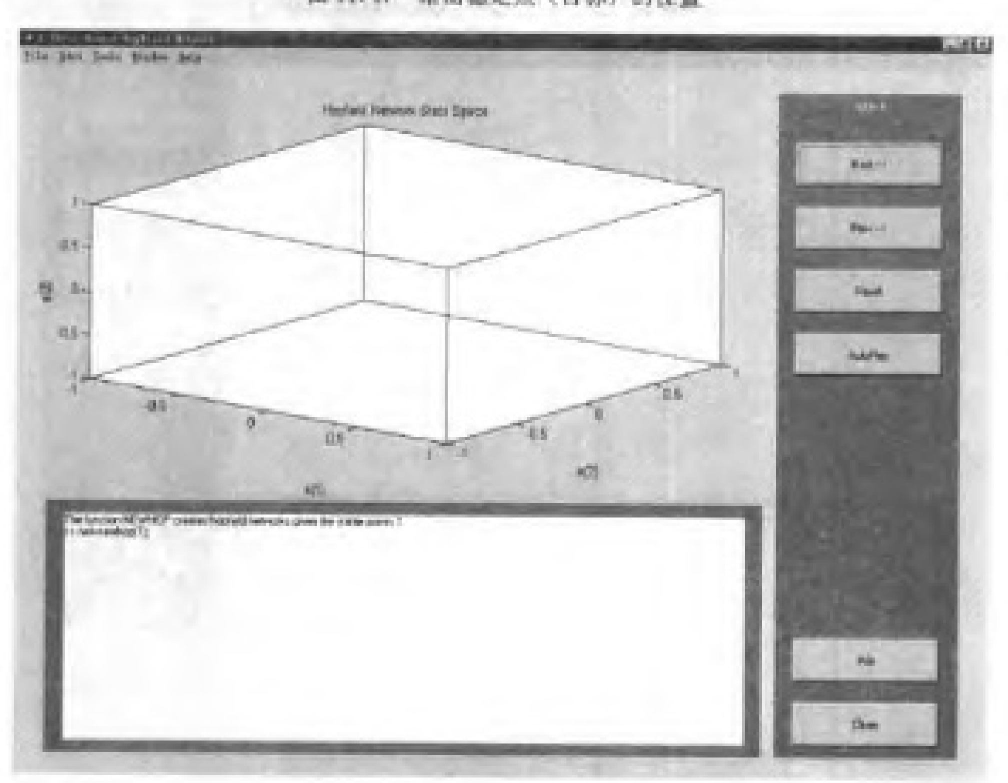


图 11.18　用稳定点（T）设计 Hopfield 网络

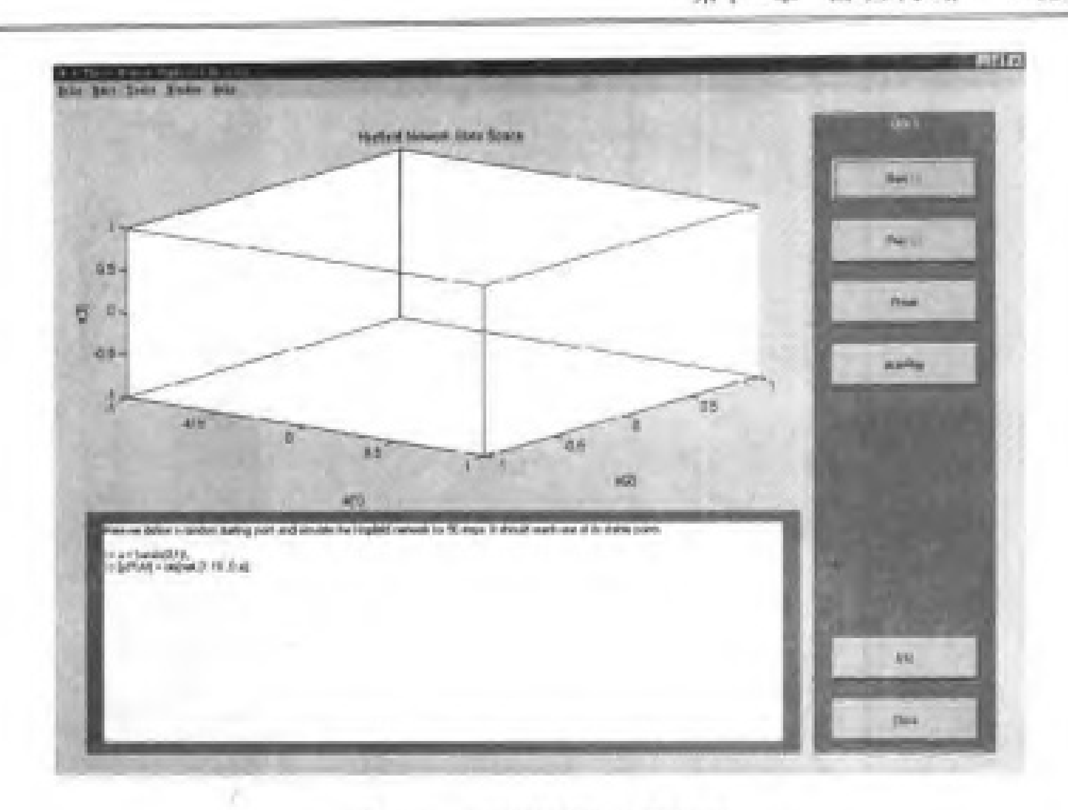


图 11.19 定义随机起始点进行仿真

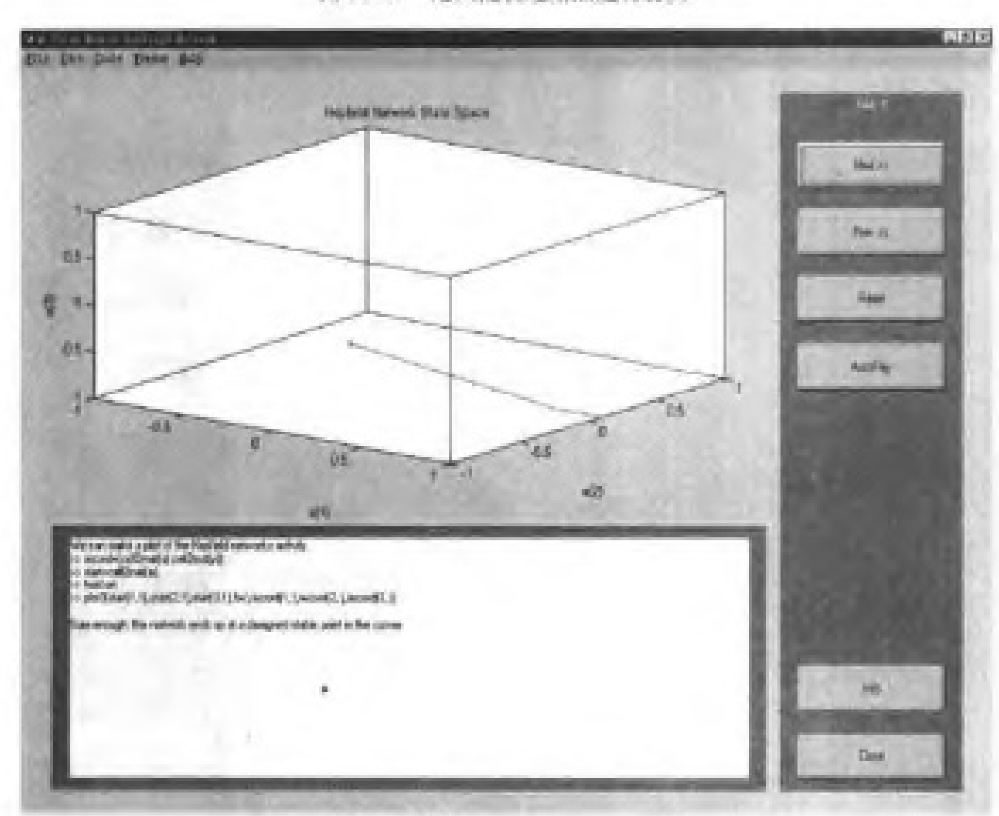


图 11.20 运行结果

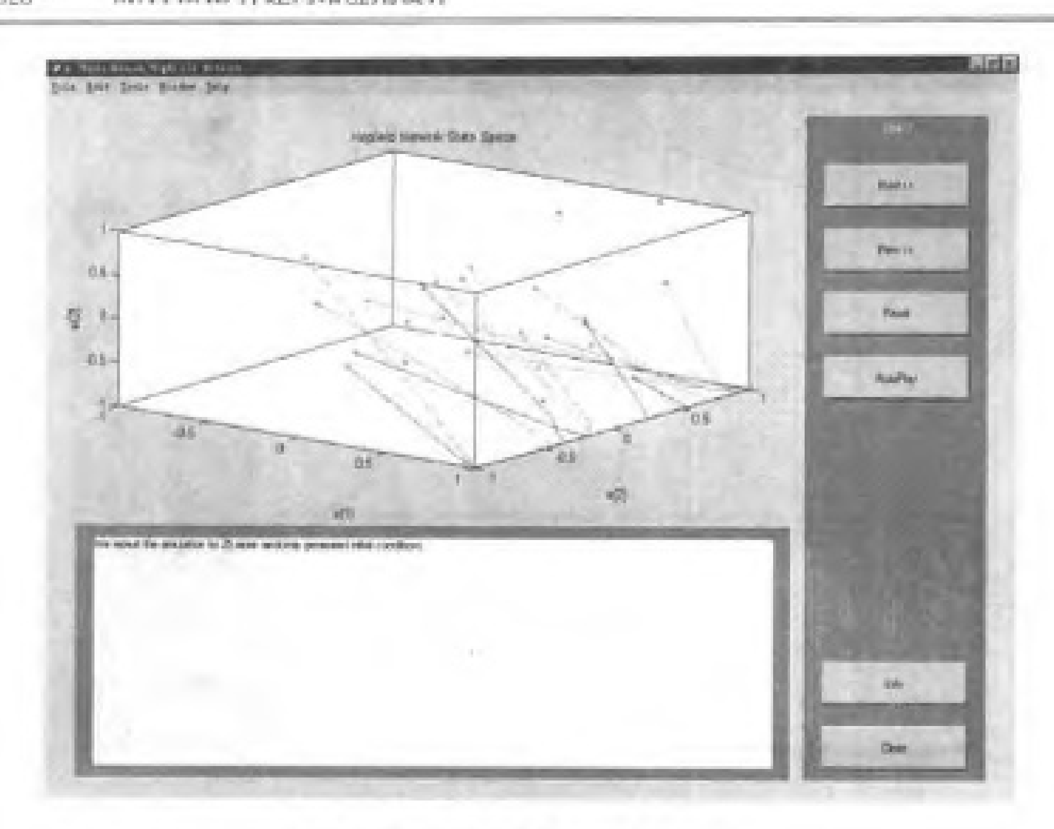


图 11.21　在 25 个初始条件下的仿真结果

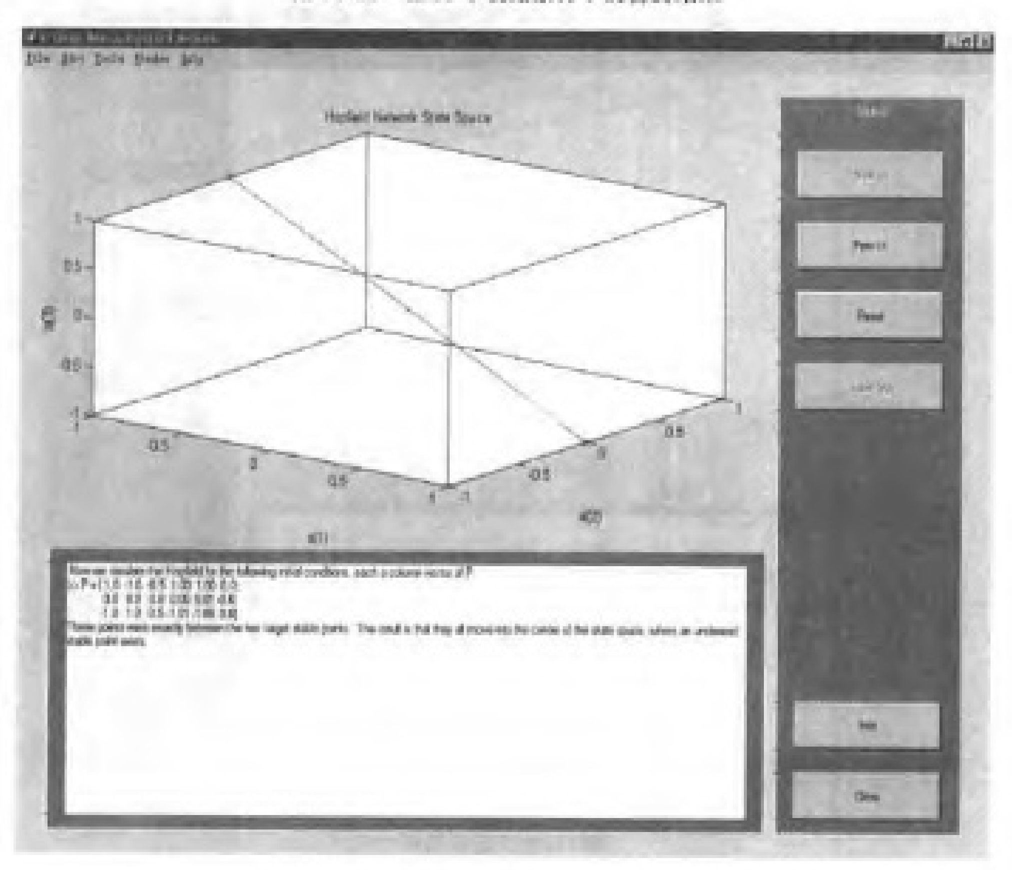


图 11.22　在图中所示的初始条件下进行仿真的结果

附录A　MATLAB（5.1版）神经网络工具箱函数

1. 基本函数

simup　感知层仿真
initp　感知层初始化
trainp　利用感知规则训练感知层
trainpn　利用规范感知规则训练感知层
simuff　前向网络仿真
initff　前向网络初始化
trainbp　用BP算法训练前向网络
trainbpx　用快速BP算法训练前向网络
trainelm　训练Elman递归网络
trainlm　用Levenberg-Marquardt算法训练前向网络
simulin　线性层仿真
solvelin　设计线性网络
initlin　线性层初始化
trainwh　用Widrow-Hoff规则训练线性层
adaptwh　用Widrow-Hoff规则自适应调节线性层的权
simurb　径向基网络的仿真
solverb　设计径向基网络
solverbe　设计精确的径向基网络
simusm　自组织映射网络仿真
initsm　自组织映射网络的初始化
trainsm　训练自组织映射网络
simuhop　Hopfield网络的仿真
solvehop　设计Hopfield网络
simuelm　Elman递归网络的仿真
initelm　Elman递归网络的初始化
trainelm　训练Elman递归网络

2. 传递函数

hardlim　硬限幅传递函数
hardlims　对称硬限幅传递函数
purelin　线性传递函数
tansig　正切S型传递函数
logsig　对数正切S型传递函数

satlin 饱和线性传递函数
satlins 对称饱和线性传递函数
radbas 径向基传递函数
dist 计算矢量间的距离
compet 自组织映射传递函数

3. 学习规则

learnp 感知层学习规则
learnpn 规范感知层学习规则
learnbp BP 学习规则
learnbpm 带动量项的 BP 学习规则
learnlm Levenberg-Marquardt 学习规则
learnwh Widrow-Hoff 学习规则
learnk Kohonen 学习规则

4. 绘图函数

plotpv 绘制具有 1/0 目标的输入矢量图
plotpc 在已存在的感知器分类图上划上分类线
plotes 绘制误差曲面
plotep 在误差曲面上绘制出权值和阈值的位置

5. δ 函数

deltalin 对 PURELIN 神经元的 δ 函数
deltatan 对 ANSITG 神经元的 δ 函数
deltalog 对 LOGSIG 神经元的 δ 函数

6. 分析函数

errsurf 计算误差曲面
maxlinlr 计算用 Widrow-Hoff 准则训练的线性网络的最大稳定学习率

附录B　MATLAB（5.3版）神经网络工具箱函数

1. 网络创建函数（new networks）

newp　创建感知器网络
newlind　设计一线性层
newlin　创建一线性层
newff　创建一前馈 BP 网络
newcf　创建一多层前馈 BP 网络
newfftd　创建一前馈输入延迟 BP 网络
newrb　设计一径向基网络
newrbe　设计一严格的径向基网络
newgrnn　设计一广义回归神经网络
newpnn　设计一概率神经网络
newc　创建一竞争层
newsom　创建一自组织特征映射
newhop　创建一 Hopfield 递归网络
newelm　创建一 Elman 递归网络

2. 网络应用函数（using networks）

sim　仿真一个神经网络
init　初始化一个神经网络
adapt　神经网络的自适应化
train　训练一个神经网络

3. 权函数（weight functions）

dotprod　权函数的点积
ddotprod　权函数点积的导数
dist　Euclidean 距离权函数
normprod　规范点积权函数
negdist　Negative 距离权函数
mandist　Manhattan 距离权函数
linkdist　Link 距离权函数

4. 网络输入函数（net input functions）

netsum　网络输入函数的求和

dnetsum 网络输入求和函数导数

5. 传递函数（transfer functions）

hardlim 硬限幅传递函数
hardlims 对称硬限幅传递函数
purelin 线性传递函数
tansig 正切 S 型传递函数
logsig 对数 S 型传递函数
dpurelin 线性传递函数的导数
dtansig 正切 S 型传递函数的导数
dlogsig 正切 S 型传递函数的导数
compet 竞争传递函数
radbas 径向基传递函数
satlins 对称饱和线性传递函数

6. 初始化函数（initialization functions）

initlay 层与层之间的网络初始化函数
initwb 阈值与权值的初始化函数
initzero 零权/阈值的初始化函数
initnw Nguyen-Widrow 层的初始化函数
initcon Conscience 阈值的初始化函数
midpoint 中点权值初始化函数

7. 性能分析函数（performance functions）

mae 均值绝对误差性能分析函数
mse 均方差性能分析函数
msereg 均方差 w/reg 性能分析函数
dmse 均方差性能分析函数的导数
dmsereg 均方差 w/reg 性能分析函数的导数

8. 学习函数（learning）

learnp 感知器学习函数
learnpn 标准感知器学习函数
learnwh Widrow-Hoff 学习规则
learngd BP 学习规则
learngdm 带动量项的 BP 学习规则
learnk Kohonen 权学习函数
learncon Conscience 阈值学习函数
learnsom 自组织映射权学习函数

9. 自适应函数

adaptwb 网络权与阈值的自适应函数

10. 训练函数（training functions）

trainwb 网络权与阈值的训练函数
traingd 梯度下降的BP算法训练函数
traingdm 梯度下降w/动量的BP算法训练函数
traingda 梯度下降w/自适应lr的BP算法训练函数
traingdx 梯度下降w/动量和自适应lr的BP算法训练函数
trainlm Levenberg-Marquardt的BP算法训练函数
trainwbl 每个训练周期用一个权值矢量或偏差矢量的训练函数

11. 分析函数（analysis functions）

maxlinlr 线性学习层的最大学习率
errsurf 误差曲面

12. 绘图函数

plotes 绘制误差曲面
plotep 绘制权和阈值在误差曲面上的位置
plotsom 绘制自组织映射图

13. 符号变换函数（signals）

ind2vec 转换下标成为矢量
vec2ind 转换矢量成为下标矢量

14. 拓扑函数

gridtop 网格层拓扑函数
hextop 六角层拓扑函数
randtop 随机层拓扑函数